IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	John A. Copeland III
Docket:	10775-36791
Title:	**NETWORK PORT PROFILING**

CERTIFICATE UNDER 37 CFR 1.10
Date of Deposit: August 9, 2005
I hereby certify that this paper or fee is being deposited with the United States Postal Service Mail Post Office To Addressee' service under 37 CFR 1.10 and is addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

By:
Name: Wendell A. Peete, Jr.

Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Customer No. 24728

Sir:

We are transmitting herewith the attached:

☒ Transmittal Sheet containing Certificate of Mailing (1 page)
☒ Request For Correction of Record (3 pages)
☒ Two Copies of Compact Disc (CD-R) labeled COPY 1 and COPY 2, entitled "Lancope Code.txt Computer Listing"
☒ Printout of Computer Program Listing on CD (73 pages)
☒ Return postcard

Please send all correspondence to:
Wendell A. Peete, Jr.
MORRIS, MANNING & MARTIN, LLP
1600 Atlanta Financial Center
3343 Peachtree Road NE
Atlanta, Georgia 30326
404-233-700 (Main)
Customer No. 24728

By:
Name: Wendell A. Peete, Jr.
Reg. No.: 52,108

(PTO TRANSMITTAL - NEW FILING)

IFW

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant:	John A. Copeland, III	Examiner:	Ronald Baum
Serial No.:	10/062,621	Docket:	10775-36791
Filed:	January 31, 2002		
Confirmation No.:	2472		
Title:	Network Port Profiling		

CERTIFICATE UNDER 37 CFR 1.10
Date of Deposit: August 9, 2005
I hereby certify that this paper is being deposited with the United States Postal Service Mail Post Office To Addressee' service under 37 CFR 1.10 and is addressed to Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

By:
Name: Wendell A. Peete, Jr.

REQUEST FOR CORRECTION OF RECORD

Customer No. 24728

Commissioner of Patents
P.O. Box 1450
Alexandria, Virginia 22313-1450

Dear Sir:

This request is being filed in order to correct the record of the currently pending application in regard to the submittal of a computer program listing saved and submitted on a first and a second compact disc (CD) in conjunction with the above-referenced patent application; said CDs apparently reported as not being received at the USPTO.

The computer program listing in question was additionally submitted on a first and second CD respectively in related United States non-provisional patent application serial no. 10/000,396 and PCT patent application PCT/US01/45,275, both entitled "Flow-Based Detection of Network Intrusions."

A status inquiry was submitted to the USPTO on March 11, 2005. The purpose of the status inquiry was to obtain an indication of when a communication would be received from the USPTO in regard to the pending application or when the pending application would be examined. Concurrently with the submittal of the status inquiry to the USPTO a search was made via PAIR of the USPTO file history. A review of the documents posted on PAIR showed that the utility patent application transmittal form submitted in regard to the above-referenced patent application had been stamped with an indicator that the USPTO had not received the CD-ROM copy one and copy two.

A review of our files indicated that the utility patent application transmittal form (attached hereto as Exhibit A) had appropriately been highlighted to specify that a CD-ROM or CD-R was attached in duplicate. A transmittal of computer program listing on CD form (attached hereto as Exhibit B) was submitted specifying therein that two copies of a CD-R compact disc were being submitted, wherein the compact discs were labeled as "Copy 1" and "Copy 2." Further, a return postcard was submitted to the USPTO, wherein conspicuously listed under the "Enclosures" section on the postcard was a statement reporting the transmittal of "Computer Program Listing on CD, copies 1 and copies 2 of a CD with program listing."

On April 20, 2005, a telephone conference was conducted with Examiner Ronald Baum, wherein the subject of the telephone conference was the non-reception of the CDs submitted with the filing of the patent application of the above-referenced matter. During the telephone conference, Examiner Baum suggested that the Applicant resubmit CD copies 1 and 2, in addition to submitting a printout of the computer code that is contained within the submitted CD-Rs.

The Applicant submits that a review of its file indicates that the CD-Rs labeled "Copy 1" and "Copy 2" were believed to have been properly submitted to the USPTO in accordance with the filing practices of Applicant's law firm. In an attempt to correct the present record of the currently pending patent application, as suggested by Examiner Baum, the Applicant hereby submits a printout of the computer code that is contained within the previously submitted CD-Rs. Further, pursuant to 37 C.F.R. § 1.52(a) the Applicant hereby submits herewith two (2) copies of one (1) compact disc (CD-R) entitled "Lancope Code.txt Computer Listing," each in conformance with the International Standards Organization (ISO) 9660 Standard, with the contents in compliance with the American Standard Code for Information Exchange (ASCII), enclosed in a hard compact case within an unsealed padded and protected mailing envelope, labeled as "Copy 1" and "Copy 2", accompanied by this transmittal letter, as required by said Rule.

In accordance with 37 C.F.R. § 1.52(e)(3)(ii), the operating system compatibility for the files contained herein are in the Microsoft Windows NT Operating System, and the CD-R includes the following ASCII text file contained on the compact disc, including the file's name, size in bytes, and date of creation, as follows:

DATE OF CREATION	SIZE IN BYTES	FILE NAME
Jan. 30, 2002	154,450	LANcope Code.txt

Pursuant to 37 C.F.R. § 1.52(e)(4), Applicant hereby states that the two compact discs identified as "Copy 1" and "Copy 2" are identical.

In accordance with 37 C.F.R. § 1.52(e)(6), each CD "Copy 1" and "Copy 2" is labeled with information indicating the name of the inventor, title of the invention, Applicant's docket number, the creation date of the compact disc (January 30, 2002), and an indication whether such copy is "Copy 1" or "Copy 2".

Applicant has made a bona fide attempt to correct the record in regard to the presently pending patent application. If any condition remains incomplete, then Applicant kindly requests that the undersigned be contacted at the address or telephone number shown below so that such condition may be met.

Respectfully submitted,

August 9, 2005

MORRIS, MANNING & MARTIN, LLP

Wendell A. Peete, Jr.
Reg. No. 52,108

MORRIS, MANNING & MARTIN, L.L.P.
1600 Atlanta Financial Center
3343 Peachtree Road, N.E.
Atlanta, Georgia 30326
404-233-7000 Main
404-495-3682 Direct
Customer No. 24728

FILE INDEX FOR FILES ACCOMPANYING THE PATENT APPLICATION OF

John Copeland
"Network Port Profiling"

Filed January 30, 2002

NOTICE OF COPYRIGHT

All files are in ASCII (text) format and saved with ".txt" file suffix. All files can be opened in Microsoft WORD 97, within WINDOWS NT.

All files are originally C code files.

Note that the "date of creation" listed below of the files is the date on which the files were created for the purpose of inclusion on this CD-ROM; the date is NOT the date on which the contents of such files were created.

There is 1 file, not including this file:

Sizein	Date	File Name
Bytes		
154,450	January 30, 2001	LANcope Code.txt

```
#define _REENTRANT    // basic 2-lines for threads
#include <pthread.h> // compile gcc -lpthread -fvolatile -fvolatile-global
#include <time.h>   //assumes service is same as port
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <signal.h>

//   lancope   000527   Thread 1 collects packet header and sorts into flows (from
ipselect.c 991215)
//   Compile: gcc lancope.c -O3 -pthread -fvolatile -fvolatile-global -o lancope
//   MUTEX order: none -> flows -> hosts -> pairs -> bs
//      You can skip a lock going to the right, but NEVER go left without unlocking
higher db's

char progid[100] = "LANcope by John Copeland 9/28/00, copyright 2000, all rights
reserved" ;
FILE  *infile, *log_flow , *xfile, *alertfile, *traffile, *config, *log_pkt_file,
*log_pair ;// files

// -------------------  CI Weights --------------- //
#define HO_ATTACK_INIT_CI  2000  //  half-open attack from initial flow eval
#define HO_ATK_PER_SYN      501  //  half-open attack per SYN, final flow eval
#define PORT_SCAN_CI        666  //  per short-term port scan detected
#define PORT_SCAN_LT_CI     999  //  per long-term  port scan detected
#define PORT_SCAN_MAX         8  //  max number or ports on all host 1st probed in
30s
#define PORT_SCAN_LT_MAX     16  //  max number or ports on all host 1st probed long
term
#define PORT_SCAN2_CI        10  //  per flow with no C or S determined
#define HI_HI_CI             50  //  per high-port to high-port connection
#define LO_HI_CI            500  //  per low-port  to high-port connection
#define LO_LO_CI            100  //  per low-port  to low-port  connection
#define UNKNOWM_CI            3  //  per low-port  to low-port  connection
#define HOST_SCAN_CI       3000  //  per address scan detected


#undef  CHECK_INDEX   // check HASH INDICES for out-of-range
#undef  TEST    //   check for URG PTR != 0, flag-exception errors
#define FLOW_DEAD  320   // inactive time -> flow finished  must be > 300 for
'persistant' http
#define LOOSE_PERIOD 900 // time in seconds for 'loose' classification of flows that
start
#define LINES_PER_SCREEN  40  // 1.6 x rough number of lines on Traffic and CI
Alerts Web screens
#define SHORT_UDP_MAX     2  //  max size for UDP data fields in a "short UDP"
#define UDP_PORT_OFFSET 2048 // added to udp ports before using port no. as offset
in port_mask[j]
#define PORT_MASK_SIZE  2058 // to dimension port_mask[]  should be 4 larger
#define LOW_PT_MAX      1024 // a "low" port is less than this

#define SLOTS  131073 //no. flows in data table > 2**SHIFT+1 8193 16385  32769 65537
131073 262145
#define SHIFT     17  // size of shifts to make index. Max =   13    14    15    16
  17    18
#define MASK  0x1ffff  //mask all but Max-SHIFT right-most bits x1fff x3fff x7fff
xffff x1ffff x3ffff
#define RANGE    15    // search range for empty slot
struct flow_db {
        unsigned long  ip[2] ; // 0 - ip address - lowest
        unsigned short pt[2] ; //tcp or udp ports
        unsigned short pt_min ; // client minimum port
        unsigned short pt_max ; // client maximum port
        unsigned long root ;
        unsigned long down ;
        unsigned long up;
        unsigned long start ;
        unsigned long last ;
        unsigned long state ; // FLOWTYPE =0,2 or for 4-TCP, 1-UDP  classified =
0x10
        unsigned short service ; // well-known port number of service, even if on a
different port
        unsigned short scans ;  // max number ports for ip0 pair
        unsigned long bytes[2] ;
        unsigned long pkts[2] ;
        unsigned long flgs[2] ;
        unsigned short binary[2]  ;
        unsigned short bin_norm[2] ;
        unsigned char flag[2][7];//0 bad, 1 reset, 2 urgent, 3 syn, 4 syn-ack, 5
fin&!ack, 6 fin&ack
        unsigned char flag_norm[2] ;// urg ptr
} flow[SLOTS] ;
// elements in flow[ ].flag[j][ FLAG ]
#define BAD 0
#define RST 1
#define URG 2
#define SYN 3
#define S_A 4  // SYN-ACK
#define ATK_CTR  // attack counter
// bits     in flow[ ].state  //  1 - udp, 2 - ip0 first, 4 - ip1 first
#define UDP_FLOW  0x01 // x0, x2, x4 are TCP stages
#define LOOSE     0x10  // loose matching for startup an split-path
#define ip0_FIRST 0x20   // ip0 sent first packet
#define ip1_FIRST 0x40    // ip1 sent first packet
#define NOT_FIRST_PKT 0x60 // first packet when neither ip0_FIRST nor ip1_FIRST is
set
#define FIRST_IN_FLO 0x60 // mask to test for "not loose and this is first pkt of
flow"
#define NO_SYN    0x80    // first TCP packet seen was not a SYN
#define CLASSIFIED 0x100  // clasified, but continue to watch
#define ATTACK     0x200  // do not continue checking (attack noted)
#define PROBE      0x400  // probe
#define ATK_PROBE  0x600  // ATTACK or PROBE
#define ATTACK_CHK 0x800  // log all packets after WATCH or ATTACK is set
#define FTP_SER_0  0x1000 // FTP_EST = FTP_OLD | FTP_PASS
#define FTP_SER_1  0x2000 // passive FTP, server is ip1 data is on a hi-hi
connection
#define FTP_ANY    0x3000 // any type FTP if true
#define HI_HI      0x4000 // both ports > 1023
#define WATCH_FLOW 0x8000 // marks flow so packets and flow is logged

#define SCAN_SLOTS 4097    // ip-pair data to detect scans,2^n + 1  1025   2049
4097    8193
#define SCAN_SHIFT   12    // size of shifts to make index. Max =n    10     11
12      13
#define SCAN_MASK 0xfff  //mask all but Max-SHIFT right-most bits    x3ff  x7ff
xfff   x1fff
#define SCAN_RANGE   25    // search range for empty slot

#define SCAN_MAX 4         // ip-pair flow[i] elements limit for same IP, different
ports

struct ip_pair {
                unsigned long ip0 ;             //ip address - source
                unsigned long ip1 ;             //ip address - target
                unsigned long root ;
                unsigned long down ;
                unsigned long up;
                unsigned long start ;
                unsigned long last ;
                unsigned long concern ;
                unsigned short n_ports ;
                unsigned short n_hits ;
                unsigned char type[16] ; // values below
                unsigned char walk[16] ; // distance of port walk
                unsigned short port[16] ;
        } scans[SCAN_SLOTS] ;

#define UDP_PROBE      1  // values for scans[i].type[ VALUE ]
#define TCP_PROBE      2
#define SHORT_UDP_SCAN 3
#define BOOMERANG      4
#define PING_SCAN      5
#define ICMP_TO        6
#define TCP_TO         7
#define UDP_TO         8
#define BAD_PKT_TRACE  9
#define TCP_PORT_SCAN 10
#define TCP_ADDR_SCAN 11
#define HALF_OPEN     12

unsigned long scin[16], scans_pr_cut, scans_cut ; // distribution of scans[].concern
values
unsigned long scan_pair(unsigned long ip0,unsigned long ip1,unsigned char
type,unsigned short port);

#define HOST_SLOTS  65537  // number Host slots           2^n + 1    16385 32769
65537 131073 262145
#define HOST_LIMIT  20000  // point to start dropping old outside host records with
CI=0 ;
#define HOST_SHIFT    16   // size of shifts to make index. Max =n       14    15
16    17    18
#define HOST_MASK  0xffff  //mask all but Max-SHIFT right-most bits   x3fff x7fff
xffff x1ffff x3ffff
#define HOST_RANGE   25    // search range for empty slot
#define HOST_MAX 4         // ip-pair flow limit HOSTs
struct host_db {
                unsigned long ip ;              //ip address
                unsigned long root ;
                unsigned long down ;
                unsigned long up;
                unsigned long start ;
                unsigned long last ;
                unsigned long udp_bytes ;  // multimedia bytes
                unsigned long bytes_in ;     // for web_alert period
                unsigned long bytes_in_pp ;  // Bytes over 5 min (wai)
                unsigned long bytes_in_mx ;  // max all day
                unsigned long pkts_in ;
                unsigned long flgs_in ;
                unsigned long bytes_ot ;     // for web_alert period
                unsigned long bytes_ot_pp    ;  // Bytes over 5 min (wai)
                unsigned long bytes_ot_mx ;  // max all day
                unsigned long pkts_ot ;
                unsigned long flgs_ot ;
                unsigned long port_smin ;  // 6 -> Server ports (0-1023)
                unsigned long port_smax ;
                unsigned long port_cmin ;   // 7 -> Client of ports ( 0-1023)
                unsigned long port_cmax ;
                unsigned long server      ;   // 32 common server ports - provided
                unsigned long client      ;   // 32 common server ports - used
                unsigned long s_profile   ;   // 32 common server ports - provided
                unsigned long c_profile   ;   // 32 common server ports - used
                unsigned short s_list[10] ;  // list of uncommon servers
                unsigned short c_list[10] ;  // list of uncommon clients
                unsigned long s_flows     ;  // Server flows
                unsigned long c_flows     ;  // Client flows
                unsigned long u_flows     ;  // Unknown (not CS) flows
                unsigned long resets      ;  //   TCP Resets
                unsigned long rejects     ;  // icmp 'port unavailable'
                unsigned long no_con_t    ;  // tcp, no response
                unsigned long dns_flows   ;  // DNS flows
                unsigned long mm_s ;  // multimedia server
                unsigned long mm_c ;  // multimedia client
                unsigned long mm_p ;  // multimedia peer
                unsigned long pt_scans ;  // no answer flows , e.g. scanning
                unsigned long host_scan[5] ;  // used to detect host scans
                unsigned char scan_cntr[4]  ;  // used to count bits port scans H32
is 0, H128 1, P_ST 2, P_LT 3
                unsigned long port_scan[4] ;  // used to detect port scans
                unsigned long bad_pkts ;   // SYN-ACK, and any other than stadard 7
                unsigned long bad_flow ;   // Not Server < 1024, Client > 1024
                unsigned long pings ;   // pings
                unsigned long traces ;  // traces
                unsigned long alerts ;  // bit map of alert condition
                unsigned long alarm_t ; // last time an alarm was set
                unsigned long concern ; // accumulated CI
        } host[ HOST_SLOTS ] ;

#define LOCAL_NETS 20  //   maximum number of local subnets
//#define IRC        0x40000   //  bits in host[].server or client above those read
in from "lancope_config"

#define LOCAL_HOST   0x1   //  bits in host[].alerts
#define PING_ALERT   0x2
#define TRACE_ALERT  0x4
#define REJECT_ALERT 0x8
#define PKT_ALERT    0x10
#define PT_SCAN_ALERT 0x20  //  from flows exceding SCAN_MAX
#define PORT_SCAN2    0X40  //  from bits in host.[h].port_scan[0]
#define HI_HI_CS      0x80
#define LO_HI_CS     0x100
#define LO_LO_CS     0x200
#define SHORT_UDP    0x400  // UDP packet with < 4 bytes data
#define IP_SCAN      0x800  // accessed too many different IP addresses
#define HO_ATTACK   0x1000  // Half-open attack
#define LT_PT_SCAN  0x2000  // Long-Term port scan
#define TRAF_ALARM  0x4000  // TRAFFIC  bytes in + out over 15 min too high.  Email
alerts sent and Web Bulletin
#define ALARM_1     0x8000  // alarm-1   talked with a suspicious host, flows & full
host info logged
#define ALARM_2    0x10000  // alarm-2  is a definite problem.  Email alerts sent
and Web Bulletin
#define NO_ALARM   0x20000  // exempt from alarms
#define WATCH_HOST 0x40000  // log packets and flows
#define NO_CS_SET  0x80000  // C-S not determined (long-duration flow?)

#define ALARM_12   0x18000  // for host[i].alarms, 1 and/or 2
#define ALARM_12W  0x58000  // alarm-1 or alarm-2  or watch-host - log this guy's
packets and flows
char alert_name[32][25] =
{"Local","Pings","Traces","Rejects","Bad_Flags","Port_Scans","Pt_Scan2",

"High-High","Low-High","Low-Low","SHORT-UDP","IP-SCAN","HO_ATTACK","LT-Pt-Scan",

"HIGH-TRAF","TOUCHED","ALARM","NO_ALM","Watch",".","","","","","","","","","","",
""  } ;

unsigned long local_net[LOCAL_NETS], local_mask[LOCAL_NETS], ln_max, app_0, app_len,
max_alert = 19 ;
unsigned long cin[2][15],
cix[15]={0,10,20,50,100,200,500,1000,2000,5000,10000,20000,50000,100000,1000000000}
;
unsigned long tin[2][15],
tix[15]={0,100,1000,10000,50000,100000,200000,500000,1000000,
2000000,5000000,10000000,20000000,50000000,1000000000} ;

unsigned long web_alert_2 = 20000, web_traf_2 = 1000000, web_traf_2byte = 37500000;
// Alarm, read from file
unsigned long web_alert0_0 = 10, web_alert1_0 = 2, web_traf0_0 = 100, web_traf1_0 =
20; // for Web
unsigned long profile, alarm_lines ;
     // web_alert_2 & web_traf_2 read in from lc_thresholds.txt, _0's adjusted by
program.
unsigned long port_mask[ PORT_MASK_SIZE ], pn_max, active_locals ; // bit map for 32
common server protocols, UDP = port + 1024
char port_name[100][32], pbuf_1[120], pbuf_2[120], serstr[400] ;         // names
for protocols above, each bit

#define TRAFFIC_VALUES 97  // no. -> 15-min intervals for 24 hours + 1
#define CLASS_FLOW_INT 30  // interval in seconds between class_flow() operations
#define WEB_ALERT_INT  300 // interval in seconds between web_alert()  operations
#define TRAF_TABLE_MIN 750 // traf table interval - (WEB_ALERT_INT / 2)

struct byte_table {
                time_t        t           ;
                unsigned long bytes_in    ;
                unsigned long bytes_out   ;
                unsigned long bytes_loc   ;
                unsigned long bytes_oo    ;
                unsigned long bytes_bc    ;
                unsigned long bytes_bad   ;
                unsigned long bytes_mcn   ;
                unsigned long bytes_mco   ;
        } bs[ TRAFFIC_VALUES ] ;

unsigned long bytes_start, spoofs, bt_old ;
int bt = -1 ;
unsigned long bps_in_max, bps_out_max, bps_loc_max, bps_oo_max  ;
unsigned long bps_mcn_max, bps_mco_max, bps_bc_max, bps_bad_max ;
unsigned long bytes_in_cnt, bytes_out_cnt, bytes_loc_cnt, bytes_oo_cnt  ;
unsigned long bytes_bc_cnt, bytes_bad_cnt, bytes_mcn_cnt, bytes_mco_cnt ;

// ###### THREADS ######## //
unsigned long thread ;
volatile time_t  t_run ;
time_t           t_zero, t_bs_last, t_wa_last, t_diff, z_sec, t_next_web, t_next_cf,
t_next_web ;
pthread_mutex_t  mp_hosts = PTHREAD_MUTEX_INITIALIZER, mp_flows =
PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t  mp_pairs = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t  mp_class_flow = PTHREAD_MUTEX_INITIALIZER  ;// for f_file
pthread_mutex_t  mp_find_flow  = PTHREAD_MUTEX_INITIALIZER ; // stops find_flow from
being reentrant
```

```
pthread_mutex_t  mp_bs         = PTHREAD_MUTEX_INITIALIZER ; // stops bs[].t from
being written while read
pthread_t        thread_num[100] ;
volatile long    np, running = 1  ;
char             monitor_start[64] ;  // capture start time

long find_slot( unsigned long ip0, unsigned long ip1, long port0, long port1); //
find slot in flow[ ]
unsigned long find_host( unsigned long ip, unsigned int make) ;
unsigned long dots_int(  char *p ) ;  // dotted-decimal (ascii) to integer
unsigned long s2i( unsigned char bb[4]) ;  // 4 bytes -> 32 bit integer, Internet
order, msb on left  [0]
unsigned long b2i( unsigned char bb[4]) ;  // 4 bytes -> 32 bit integer,
intel-86order,  msb on right [3]
void    print_host(FILE *xfile, unsigned long i, unsigned long x) ;
void usage( void ) ;
int start_tcpdump( void ) ; // returns '0' if ok
void kill_tcpdump( void ) ;
void process_pkts( void ) ;  // run as threads
void class_flow( void ) ;
void web_alerts( void ) ;
void record_probe( unsigned long hs, unsigned long des_addr, unsigned short des_port
) ;

void services( unsigned int x ) ;
void read_thresholds( void) ;
int  read_profiles( void ) ;
int  save_profiles( void ) ;
void suicide( int yesterday) ; // saves logs and restarts this program (or newer
version)
void userhandler(int) ;  // SIGTERM handler (control-c response)
#ifdef CHECK_INDEX
        unsigned long host_hash( unsigned long ip) ;
#endif

unsigned char snp_hdr[40], snp_buf[40], lan_buf[40], ip_buf[3000], *tcp_buf ;
//buffers
unsigned char  ct[80], rt[80], as[20] ;
unsigned int           n_host = 1, n_pr_search, n_scans, slots_cut, slots_pr_cut ;

//      FILE *tf ; //DeBuG    unsigned long temp[128] ;

unsigned long   i_src, i_des,  hs_ip[128] ;

int ip_options_0 = 20, ip_options_1, data_0, data_1, restart_hour = 6 ; // gmt
long n_miss, n_max, n_flow, n_flow_log, t_read ;
unsigned char transport, scan_max ;
unsigned int  des_addr, src_addr, src_port, des_port, ah, tcp_all ;
unsigned short  err_limit = 40, n_restart ;
unsigned long   flag_alr[256],  flag_cnt[256], flags, n_icmp, n_tcp, n_udp, n_full,
port  ;
unsigned long   snp_rec_max , host_cut ;  // 24 + snaplen option used for Snoop

int  f_dotdec, f_all, f_encrypt, f_file, f_demo, f_splitpath, f_loose, f_verbose,
f_noread ;  // flags
int  n_active, n_active_max ;  //  used by class_flow()
int n_lost ;  // flows lost due to no slot
int            lan_hdr, both_hdr, snp_len, snoop ;// lan_hr = 14 Ethernet, 21 FDDI
unsigned long    non_ip, ip_not4, short_ip, ip_len[16], not_tui ; // for cleaning


int main(int argc, char * argv[ ]) //    ============= MAIN ==================
{
        struct tm       *s_time, *cap_time  ;
    clock_t     t_1, t_2  ;  // t_1 = time in 1/60 s ,     clock_t is unsigned long
    time_t      t_now, snoop_t ;     // time in sec since 1-1-1970, unsigned long
  struct timespec ts ; // ts.tv_sec and ts.tv_nsec
  char  *inp, logflo[130], logpkt[130], logscan[130], *opx,  app[130], alt[130] ;//
inp:infile
  unsigned long i, j, k, h, n, index, index0 ;          //  index to data table
        unsigned long   pkt_len, inclu_len, rec_len, cu_drop ;  //  pkt record
header
        unsigned long   t_start, t_last ;  /* time */
        unsigned char    x, y ;
        unsigned long     save_old, n_list, n_err, m ;  //  used while reading
// --------------- FOR HOST STATISTICS ------------------
        int   ntf, nprof, hdr_needed ;
        unsigned short port_ck = 0 ;  //  TCP or UDP port numbers
        unsigned char c, flg , run_flags[120];
//  Unique to Select.c
        unsigned long ipv[40], n_ips, v, snap_len, cap_len ;
        char ips[200], *sp1, *sp2, *cap_prog, as0[200], as1[200], cmd[ 100 ],
filebase[100]  ;
        unsigned long  ip0, ip1, n_slot, no_header ;
        void  (*handlerptr)(int); // for SIGTERM handle

        printf("\nProgram ID: %s\n   ptr host[1] : %u, %u\n" , progid, (size_t)
&host[2],
        (size_t) &host[2].ip ) ;
        if ( argc < 3 ) {  //  Snoop Pkt Size optional
                printf(" *** Not Enough Auguments\n"); usage() ;
        }

        handlerptr = signal(SIGTERM, userhandler);
        if (handlerptr == SIG_ERR)      printf("Can't assign signal handler.\n");

                inp = argv[1];  //input file name

//      if(strstr(inp,"/")) {
//              printf("\n  ### INPUT FILE MUST BE IN THE DEFAULT DIRECTORY
###\n\n");
//              usage() ;
//      }

        if(argc > 2) strcpy(logflo,argv[2]) ;

//      tf  = fopen("host_scan_data.txt","w") ;  // temp - Print Host Scan Data //
DeBuG

        t_now = time( NULL ) ; // present time in seconds

        if(logflo[0] == '-') {
                unsigned long day, hour ;
                day  = (t_now / 86400) % 30 ;
                hour = (t_now /  3600) % 24 ;
                if( hour >= restart_hour) day = (day + 1) % 30 ;
                sprintf(logflo,     "log-flows-%u.txt" , day ) ;  // use same files
if restart in same day
                sprintf(logpkt, "log-pkts-%u.snp"  , day ) ;
//              struct tm               *w_time ;
//              w_time = localtime( & t_now ) ;  // w_time points to 'tm' structure
//      strftime(logflo, 16, "%y%m%d%H", w_time) ; // t_table is formatted into
string ts

        }
//      strcpy( filebase, logflo ) ; // save for restart
//       strcpy(logpkt, logflo) ;
//       strcat(logpkt, "log.snp" ) ;
//       strcpy(logscan, logflo) ;
//       strcat(logscan, "-scans.snp" ) ;
//       strcat(logflo, "-flow.txt" ) ;

printf("Selected flows listed in File: '%s', packets in '%s'\n", logflo, logpkt) ;

#ifdef CHECK_INDEX
        printf("    --- INDEX CHECKING IS TURNED ON  ---    \n");
#endif

        if(argc > 3) {   //  --- set flags
                strcpy(run_flags,argv[3]) ;
                printf(" Flags: '%s',", run_flags );
//              if(strstr(run_flags,"d")) {f_dotdec  = 1; printf(" dotdec, "  ) ;
}// write IP addrs in dotted decimal
                if(strstr(run_flags,"a")) {f_all     = 1; printf(" all flows,") ; }
// fprint all flows that terminate
                if(strstr(run_flags,"e")) {f_encrypt = 1; printf(" encrypt, " ) ; }
// encrypt local net names
                if(strstr(run_flags,"f")) {f_file    = 1; printf(" file in, " ) ; }
// take t_pace from file
                if(strstr(run_flags,"o")) {f_demo    = 1; printf(" demo, "      );}
//  demo, list all hosts
                if(strstr(run_flags,"n")) {f_noread   =1; printf(" no profiles, ");}
//  skip old profiles
                if(strstr(run_flags,"s")) {f_splitpath=1; printf(" split_path, ");}
//  some packets missing
                if(strstr(run_flags,"v")) {f_verbose  =1; printf(" verbose, "   );}
//  verbose
                if(strstr(run_flags,"vv")) {f_verbose =2; printf("-very" );} //
very verbose
                if(strstr(run_flags,"vvv")){f_verbose =3; printf("-very" );} //
very verbose
                if(strstr(run_flags,"vvvv")){f_verbose=4; printf("-very" );} //
very verbose
                printf("\n");
        }
        f_loose = 1 ; // after t_run > LOOSE_PERIOD, f_loose = f_splitpath

// ----------------   File opening for input from TCPDUMP or real file---------- //
        if(f_file == 0 ) {
                if( start_tcpdump() ) {
                        printf(" ### eth1 OR tcpdump CAN NOT BE STARTED -
TERMINATING ###\n\n");
                        usage() ;
                }
        }
        else {
                if((infile = fopen (inp,"rb"))==NULL) {   //  OPEN INPUT FILE//
      printf("Input File %s Can Not Be Read\n", inp) ;
        usage() ;
                }
        }
        if((config = fopen ("lancope_config.txt","rb"))==NULL) {   //  OPEN INPUT
FILE
      printf("Input File 'lancope_config.txt' Can Not Be Read\n", inp) ;
     usage();
        }
        while(1) {  // read configuration parameters - port numbers,mask, and names
                unsigned int i , j ;
                if( EOF == fscanf(config," %u %u", &i, &j)      ) break ;  // read
```

```
values of i, j
                if(( i >= 9000 ) || ( j > 31 )) break ;
                if( EOF == fscanf(config," %s", &port_name[j] ) ) break ;  // read
port name
                if( i <= ( 2 * UDP_PORT_OFFSET )) {
                        port_mask[ i ] = 1 << j ;
                        pn_max = j + 1 ;
        }       }
        port_mask[ 2 * UDP_PORT_OFFSET + 1 ] =  1 << pn_max ;
strcpy(port_name[pn_max], "MMedia"); pn_max ++;
        port_mask[ 2 * UDP_PORT_OFFSET + 2 ] =  1 << pn_max ;
strcpy(port_name[pn_max], "M'cast"); pn_max ++;
        port_mask[ 2 * UDP_PORT_OFFSET + 3 ] =  1 << pn_max ;
strcpy(port_name[pn_max], "B'cast"); pn_max ++;
        port_mask[ 2 * UDP_PORT_OFFSET + 4 ] =  1 << pn_max ;
strcpy(port_name[pn_max], "Odd:")  ; pn_max ++;

        ln_max = 0 ;
        while(1) {  // ###  read configuration parameters - local sub-nets ###
                if( EOF == fscanf(config," %s %s", pbuf_1, pbuf_2) || (atoi(pbuf_1)
> 256)) break ;
                local_net[  ln_max ] = dots_int( pbuf_1 ) ;  //  dotted decimal to
integer
                local_mask[ ln_max ] = dots_int( pbuf_2 ) ;
                if( f_encrypt ) {
                        unsigned int a ;
                        a = local_net[ ln_max ] ;
                        if( ! (local_mask[ ln_max ] & 0x0f00 )) local_mask[ ln_max ]
&= 0xffff0000 ; //for B-C class
                        local_net[ ln_max ] ^= (((a>>4)&0x0f0f0000) ^
(a<<4)&0x0000f000) ; //encrypt routine 'c'
                        a = local_net[ ln_max ] ;
                        printf("Encrypted Local Net: %u.%u.%u.%u, \t", a>>24,
(a>>16) & 0xff, (a>>8) & 0xff, a & 0xff ) ;
                }
                if( (++ln_max) >= LOCAL_NETS ) break ;
                printf("Local Net: %15s, Net Mask: %15s\n", pbuf_1, pbuf_2 ) ;
        }
        fclose( config) ;

        if((log_pair = fopen("log_pair.txt","ab"))==NULL) {
        printf("\nLog-Packets File 'log_pair.txt' Can Not Be Opened for Append\n\n")
;
        usage() ;
        }

pthread_mutex_lock( &mp_hosts) ;    //  ### lock hosts[]
        read_thresholds() ; // initial values for web_alert2, web_traf2,
restart_hour, profile ;
pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]  - no_alarms and
watch_list


        if( ( ! f_noread) && ( profile >> 0 ) ) read_profiles() ; // ###  read
profiles ###
        printf("\n Waiting for IP Data\n") ;

// ---  read SNOOP FILE header into snp_buf[ ]
        no_header = 1 ;
        while( no_header ) {
                no_header = 0 ;
                n = fread( (void *) snp_hdr, 4, 4, infile) ;
        if( n<4) {
            printf("Could not read file header from %s\n\n", inp) ;
            no_header = 1 ;
        }
        else {
            if(
(snp_hdr[0]=='s')&&(snp_hdr[1]=='n')&&(snp_hdr[5]==0)&&(snp_hdr[11]==2) ){ // snoop

                snp_hdr[7] = 0x00 ;
                printf("%s %s Version %u, Format %u (4=Ethernet,
8=FDDI)\n", snp_hdr, &snp_hdr[6], \
              (int) snp_hdr[11], (int) snp_hdr[15]);

                if( snp_hdr[6] == 'c' ) f_encrypt = 1 ; // encrypt
local net addresses  to match file
                if( snp_hdr[15] == 4 )  lan_hdr = 14 ; // Ethernet
LAN - SNOOP file
                else {
                    if(snp_hdr[15] == 8 ) lan_hdr = 21 ; // FDDI
LAN
                    else {
                printf("\nFile is not 'snoop' of Ethernet or FDDI.
Type = %d\n\n", (int) snp_hdr[15] ) ;
                        no_header = 1 ;
                }       }
                both_hdr = lan_hdr + 24 ;

                if(
(snp_hdr[0]!='s')||(snp_hdr[1]!='n')||(snp_hdr[5]!=0)||(snp_hdr[11]!=2) ){
                        printf("\nFile is not 'snoop'
version '2' \n\n") ;
                    no_header = 1 ;
                }
                cap_prog = "snoop" ;
            //  Read 1st Snoop Record Header to get starting time
                n = fread( (void *) snp_buf, 4, 6, infile) ;    //
read  1st  snoop RECORD header into pf[ ]
                if( n < 6) { printf("\nEOF BEFORE FIRST RECORD\n\n")
;usage() ; }

            //  pkt_len   = s2i( &snp_buf[ 0] ) ;
            //  inclu_len = s2i( &snp_buf[ 4] ) ;  // captured
length
                rec_len   = s2i( &snp_buf[8]) ;    // SNOOP record
length (24 + captured length)
            //  cu_drop   = s2i( &snp_buf[12] ) ;  // not used
                t_read = snoop_t = s2i( &snp_buf[16] ) ;  //
absolute capture time in seconds
                snoop = 1 ; // cap program was snoop

            } else {                        // tcpdump

                if(  snp_hdr[3] < 0xa1 ){
                    printf("\nFile is not 'snoop' version '2' or
a TCPDUMP -w file\n\n") ;
                    printf("1st 4 bytes are: %x,%x,%x,%x\n",
snp_hdr[0],snp_hdr[1],snp_hdr[2],snp_hdr[3]);
                    usage() ;
                }
                lan_hdr = 14 ; // Ethernet
                both_hdr = lan_hdr + 24 ;
                cap_prog = "tcpdump" ;
                snoop = 0 ;  // not snoop
```

```
                                n = fread( (void *) &snp_hdr[16], 4, 2, infile) ;
// get extra length of tcpdump file header
                                if( n < 2) {
                                        printf("\nEOF BEFORE FIRST RECORD\n\n") ;
                                         no_header = 1 ;
                                }
                                else {
                                        snap_len = snp_hdr[16] + 256 * snp_hdr[17] ;

                                //  Read 1st tcpdump Record Header to get starting
time
                                        n = fread( (void *) snp_buf, 4, 6, infile) ;
// read  1st  snoop RECORD header into pf[ ]
                                        if( n < 6) {
                                                printf("\nEOF BEFORE FIRST
RECORD\n\n") ;
                                                no_header = 1 ;
                                        }
                                        else {
                                                rec_len   = b2i( &snp_buf[8]) + 24 ;
   // tcpdump record length (24 + captured length)
                                                t_read = snoop_t   = b2i(
&snp_buf[0] ) ;       // absolute capture time in seconds
                                                printf(" tcpdump -s (raw) capture
file, snap_len = %u\n", snap_len ) ;
                                        }
                                }
                        }  // end tcpdump specific read
                }


                if( no_header ) {
                        if( f_file ) usage() ;
                        else  {
                                start_tcpdump() ;
                                printf("  #### WILL TRY TO READ HEADER AGAIN ###
\n") ;
                }       }
                if( running != 1 ) goto wrapup ;
                sleep(1) ;
        } // end - while( no_header)

        snp_rec_max =250 ;

//      printf("Run_flags: '%s', Snoop Record Max. Length: %u bytes\n", run_flags,
snp_rec_max ) ;

        {
        struct stat file_info;   //  ---------------OPEN OUTPUT FILE log-flow-n.txt
and log-pkt

                if(  stat( logflo, &file_info) != 0 ) {       // file logflo does
not exists already
                        if((log_flow = fopen (logflo,"wb"))==NULL) {
                        printf("\nLog-Packets File %s Can Not Be Created\n\n",
logflo) ;
                        usage() ;
                        }
                        hdr_needed = 1 ;
                }
                else {                                           // file logflo
exists already
```

```
                        if((log_flow = fopen (logflo,"ab"))==NULL) {
                        printf("\nLog-Packets File %s Can Not Be Opened for
Append\n\n", logflo) ;
                        usage() ;
                        }
                        hdr_needed = 0 ;
                }

                if(  stat( logpkt, &file_info) != 0 ) {   // file logpkt does not
exists already
                        if((log_pkt_file = fopen (logpkt,"wb"))==NULL) {
                        printf("\nLog-Packets File %s Can Not Be Created\n\n",
logpkt) ;
                        usage() ;
                        }
                        fwrite( snp_hdr, sizeof(char), 24, log_pkt_file ) ; // copy
tcpdump file header into packet log file
                }
                else {                                              // file logpkt
exists already
                        if((log_pkt_file = fopen (logpkt,"ab"))==NULL) {
                        printf("\nLog-Packets File %s Can Not Be Opened for
Append\n\n", logpkt) ;
                        usage() ;
                        }
                }
        }

        t_now =  t_1 = time(NULL) ;   //  puts present time (s) into t_now and
GLOBAL t_zero
        if( f_file) { t_zero = t_read ; } else { t_zero = t_now ; }
        s_time = localtime(&t_now);     // puts ascii data in struct( *captime),
makes s_time point to it
        strcpy( rt, asctime( s_time ) ) ; // rt  has \n

        cap_time = localtime( &snoop_t ) ;
        strcpy( ct, asctime( cap_time ) ) ;
        strcpy(monitor_start, "Monitoring Started: ") ;
        strcat(monitor_start, ct ) ;
        monitor_start[ 39 ] = ',' ; // chop off year and \n for Web page title
        monitor_start[ 40 ] = ' ' ;
        monitor_start[ 41 ] =  0  ;

        fflush(stdout) ;
        t_2 = clock() ;

        printf("Data captured by %s to file %s on  %s", cap_prog, inp,  ct ) ; // ct
has \n
        printf("Program ID: '%s'.\nRun on %s.  Output file: %s\n", progid, rt,
logflo);
        if( hdr_needed ) {
fprintf(log_flow, "Data captured by %s to file %s on  %s", cap_prog, inp,  ct ) ; //
ct has \n
fprintf(log_flow, "Program ID: '%s'.\nRun on %s.  Output file: %s\n", progid, rt,
logflo);
fprintf(log_flow,"prot\tip0\tip1\tport0\tport1\tservice\tstart\tlast\tbytes0\tpkts0\
tflags0\t");
fprintf(log_flow,
"p-min0\tp-max0\tbinary0\tBad0\tRST0\tURG0\tSYN0\tSYN-ACK0\tFIN0\tU-PTR0\t" );
fprintf(log_flow, "bytes1\tpkts1\tflags1\tp-min1\tp-max1\t");
fprintf(log_flow, "binary1\tBad1\tRST0\tURG1\tSYN1\tSYN-ACK1\tFIN1\tU-PTR1\n" );
        }
        // #####################  START THREADS TO READ PACKETS and CLASSIFY FLOWS
```

```
#######

        pthread_create( &thread_num[1], NULL, (void *) &process_pkts,  NULL ); //
### NEW THREADS
                if( f_file )                            pthread_mutex_lock(
&mp_class_flow ) ; // block  class_flow() until p_p releases
        pthread_create( &thread_num[2], NULL, (void *) &class_flow   ,  NULL );
//      pthread_create( &thread_num[3], NULL, (void *) &web_alerts ,  NULL ); //
ALERT TABLES FOR WEB

// ### Three threads run until  userhandler()  detects SIGTERM and sets 'running =
2' ###

        pthread_join( thread_num[1], NULL ); // wait for completion - until
'running' -> 2
        printf(" Thread-1 (process_pkts) joined, ");
        running = 2 ;
if( f_file)  pthread_mutex_unlock( &mp_class_flow ) ;
        pthread_join( thread_num[2], NULL ) ;
        printf(" Thread-2 ( class_flow ) joined, ");
//      pthread_join( thread_num[3], NULL ) ;
//      printf(" Thread-3 ( web_alerts ) joined\n");

        // ##################### END OF THREADS ####################

wrapup:

        printf("\n*** Packet Data Source Ended, Flow Log: %s, Pkt Log:
%s\n\n",logflo, logpkt );
        time(&t_now) ;
        s_time = localtime(&t_now) ;

        xfile = stdout ; // 1st time through
                fprintf(xfile," Flows found: %u, logged: %d,  packets: %u, time-out:
%d \n",
                                                  n_flow, n_flow_log, np, FLOW_DEAD)
;
                fprintf(xfile," TCP packets: %u, UDP packets: %u, ICMP packets:
%u\n", n_tcp, n_udp, n_icmp) ;
                fprintf(xfile,"\nIt took %u seconds to process, %u pkts, %u cap.
secs.\n",time(NULL) - t_1,np, t_run);
//              xfile = log_flow ; // 2nd time through
//{
//      for( i = 0 ; i < 128 ; i++ ) {                // DeBuG
//              fprintf(tf, "%u\t%u\t", i, temp[ i ] ) ;
//              if(hs_ip[ i ] == 0 ) fprintf(tf, "\n") ;
//              else {
//                      register unsigned int a ;
//                      a = hs_ip[ i ] ;
//                      fprintf(tf, "%3u.%3u.%3u.%3u\n", a>>24, (a>>16) & 0xff,
(a>>8) & 0xff, a & 0xff) ;
//              }
//      }
//      printf("  Host Scan Data saved in file host_scan_data.txt\n");
//}
        save_profiles() ;

        for( i = 1 ; i < SCAN_SLOTS ; i++ ) { //    ### WRITE SCAN-PAIRS  ###
IDENTICAL TO CODE IN web_alerts()
                register unsigned int ip0, ip1, done;
                char as0[20], as1[20], as[30], as4[30], pr_str[500], x_str[50] ;
                if( scans[i].down)  { //  select pairs to fprint
                                ip0 = scans[i].ip0 ;
                                ip1 = scans[i].ip1 ;
        sprintf( as0, "%3u.%3u.%3u.%3u",(ip0>>24) & 0xff, (ip0>>16) & 0xff, (ip0>>8)
& 0xff, ip0 & 0xff) ;
        sprintf( as1, "%3u.%3u.%3u.%3u",(ip1>>24) & 0xff, (ip1>>16) & 0xff, (ip1>>8)
& 0xff, ip1 & 0xff) ;
                                for( j = 0 ; j < strlen(as0) ; j++ ) if( as0[j] == '
') as0[j] = '0' ; //  target ip0
                                for( j = 0 ; j < strlen(as1) ; j++ ) if( as1[j] == '
') as1[j] = '0' ;  // prober ip1

                sprintf(pr_str, "%u\t%u\t%s\t%s\t%u\t%u\t", scans[i].last,
scans[i].start, as0, as1,
                   scans[i].n_hits, scans[i].n_ports );
                        for( j = 0 ; j < 16 ; j++ ) {
                                x_str[0]  = 0 ;  done = 0 ;
                                switch ( scans[i].type[j] ) {
        case  1:  sprintf(x_str, "UDP_R-%u", scans[i].port[j])      ;break ; //
UDP_PROBE
        case  2:  sprintf(x_str, "TCP_R-%u", scans[i].port[j])      ;break ; //
TCP_PROBE
        case  3:  sprintf(x_str, "Short_UDP-%u" , scans[i].port[j]) ;break ; //
SHORT_UDP_SCAN
        case  4:  sprintf(x_str, "Src=Des-%u", scans[i].port[j])    ;break ; //
BOOMERANG
        case  5:  sprintf(x_str, "Ping")                            ;break ; //
PING_SCAN
        case  6:  sprintf(x_str, "ICMP_TO" )                        ;break ; //
ICMP_TO
        case  7:  sprintf(x_str, "TCP_TO-%u", scans[i].port[j])     ;break ; //
TCP_TO
        case  8:  sprintf(x_str, "UDP_TO-%u", scans[i].port[j])     ;break ; //
UDP_TO
        case  9:  sprintf(x_str, "Bad_TCP-%u", scans[i].port[j])    ;break ; //
BAD_PKT_TRACE
        case 10:  sprintf(x_str, "Multi_Pt-%u", scans[i].port[j])   ;break ; //
TCP_PORT_SCAN
        case 11:  sprintf(x_str, "Addr_Scan-%u", scans[i].port[j])  ;break ; //
TCP_ADDR_SCAN
        case 12:  sprintf(x_str, "HO_ATTACKn-%u", scans[i].port[j]) ;break ; //
HALF_OPEN
        default : done = 1 ;
                                        }
                                if( done ) break ;
                                strcat( pr_str, x_str ) ;
                                if( scans[i].walk[j] > 0 ) {
                                        sprintf(x_str, "(%u) ",  scans[i].walk[j] )
;
                                        strcat( pr_str, x_str) ;
                                }
                                else  strcat(pr_str, " " ) ;
                        } // end j = 0, 1,2, ...
                        strcat(pr_str, "\n") ;
                }
                fprintf(log_pair,"%u\t%s",scans[i].concern, pr_str) ;  // write to
log before deleting
        }

        n = 0 ;
        for( i = 1 ; i < SLOTS ; i++ ) {
                if(flow[i].down) {
                        n++;
                }
        }
        printf("  Max. Search: %d, Max. Flows Active: %d, Still Going: %u\n", n_max,
n_active_max, n );
        kill_tcpdump() ;

//      running = 2 ;  // finish up - force all flows to terminate
//  class_flow() ;
//  web_alerts() ;

//      printf("  All Flows Terminated and Classified\n");

//      sprintf( cmd, "lc_restart  %s  %s &", argv[3], filebase ) ;
        exit(0) ;
}               //   ++++++++++++++++++++++++  END MAIN()  ++++++++++++++++++++++


void process_pkts( ) //   ### THREAD #1 ###
{
        unsigned long i, j, k, h, index, index0 ;               //  index to data
table
        unsigned long   pkt_len, inclu_len, rec_len, cu_drop ;  //  pkt record
header
        long m, c_sec, t_sec, t_us, t_offset=0, t_cf, t_start ;  // t_offset + or -
        unsigned long  ip0, ip1, n_slot, n_err, n,  h0, h1, hs, hd, bytes, bytes_hdr
;
        time_t t_now, t_check, t_pause = 3 ;

        t_now = time(NULL) ;
        z_sec = t_read  ;       // snoop time stamp values
        c_sec =  z_sec ;
        t_run = 1 ;             //  seconds since capture run started, must be > 0
        t_start  = t_cf = 0 ;
        non_ip = ip_not4 = short_ip = n_err = not_tui = 0 ;
        for( i = 0; i < 16;  i++) ip_len[i] = 0 ;  // no. IP frames with header
length i,  i >< 5


 //     32  16   8   4   2   1
        for( i = 1 ; i < 256 ; i++ ) {  // x,x,URG,ACK ,PSH,RST,SYN,FIN
                if( ( i & 0x3f ) == 0x3f ) flag_alr[i] |= 0x10000 ; // Christmas
attack
                if( ( i & 3  )   ==    3 ) flag_alr[i] |= 0x20000 ; // Syn Fin,
Jackel or nmap
                if( ( i & 0xc0 ) >   0   ) flag_alr[i] |= 0x40000 ; // Reserved
Flags
                if( ( i & 4   ) && ( i & 0x23)) flag_alr[i] |= 0x080000 ; // RST and
any but PSH, ACK
                if( (~i & 0x10) && ( i & 0x29)) flag_alr[i] |= 0x100000 ; // no ACK,
any flag but SYN or RESET
                if( ( i & 0x20) && ( i & 0xc7)) flag_alr[i] |= 0x200000 ; // UGR +
any except ACK or PSH
        }
        flag_alr[0] = 0x400000 ; // no flags
        np = 0 ;

        if( f_file ) {
                unsigned int cfi = CLASS_FLOW_INT ;  // cfi = 30 s
                unsigned int wai = WEB_ALERT_INT  ;  // wfi = 900
                t_next_cf = *(&t_zero) + 0.5 * cfi ;
                t_next_cf = t_next_cf + cfi - (t_next_cf % cfi) ; // moves to
nearest even value
        }
```

```
    restart_here:   printf("# Thread process_pkts start (%d) at t: %u, np:
%u\n", n_restart++, t_run, np) ;

    while (running == 1) {    // ############### START LOOP TO READ PACKETS
###############
        long port0, port1, log_pkt ;
        if( np == 0 ) goto read_ip_buf ;  // already read snoop header
 // *** PKT TIME FROM SNOOP HEADER ****
        if( f_file) {
            t_sec = t_read + t_offset;
            if (((( t_sec - c_sec) > 360 ) || ( t_sec - c_sec) < -10 )
|| ( t_sec < z_sec  )) {
    //              printf( "BAD TIME VALUE, np:%u, Chk. T = %d, This T
= %d s, %d us, Error = %u",\
                        np, c_sec, t_sec, t_us, n_err );
    //              if( abs(c_sec - t_sec) > 120 ) {  //   clock jump
too large, ignore
    //                  printf(" - Time Jump > 120 s. PACKET
IGNORED.\n") ;
    //                  goto read_next ;
    //              }
    //              else printf("\n");
                    printf("np: %u, Old T Offset = %d,  ", np, t_offset
) ;
                    t_offset += c_sec - t_sec ;
                    t_sec = c_sec ;
                    printf( ", New T Offset =  %d\n", t_offset, t_run )
;
                    if ( n_err++ > 30 )  break ;   // go to wrapup
            }
            c_sec = t_sec ;
            t_run = t_sec - z_sec + 1 ;   // data from file
        } else {
            t_run = t_read - z_sec + 1 ; // real-time, z_sec is time of
first packet
        }
// ============================    PROCESS PACKET
======================================
        log_pkt = 0 ; // do not log packet unless this is set
pthread_mutex_lock( &mp_flows) ;  //  ### lock hosts[]
pthread_mutex_lock( &mp_hosts) ;  //  ### lock hosts[]
if( f_verbose == 3 ) { printf(" np:%u received at %u, ", np, t_run ) ;
fflush(stdout)  ; }
        if( !( (lan_buf[lan_hdr - 2] == 8) && ( lan_buf[lan_hdr - 1] == 0) )
) {  // not an IP packet
            non_ip++ ;
            goto read_next ;
        }
        if( ( ip_buf[0] & 0xf0 ) != 0x40 ) {  //IP, but not version 4
            ip_not4++ ;
            goto read_next ;
        }

        if( ( n = (ip_buf[0]) & 0x0f ) != 5 ) ip_len[ n ] ++ ; //IP, but
with options

// ---------IP actions ------
        m = 4*n ;   //  zero data, start i adjusted for length of IP header
        if(( m < 0 ) || (m > 32)) {
            n_err++ ;
            printf("Length-Error Packet, %u.  Length = %d 4-byte
words.\n", np, n) ;
            log_pkt = 1 ;
```

```
//                          if( n_err > err_limit ) goto wrapup ;
//                          else
                                    goto read_next;
                    }
                    tcp_buf = &ip_buf[ m ] ;  // ip header length m = 20 unless there
are IP options

//                  ip_options_1 =  m ;

                    src_addr  = s2i( &ip_buf[12] ) ;
                    des_addr  = s2i( &ip_buf[16] ) ;
                    transport = (unsigned char) ip_buf[ 9 ] ;

                    hs = find_host( src_addr, 1 ) ;  // create entry if none exists
                    hd = find_host( des_addr, 0 ) ;  // find hd if entry exists

                    if(src_addr == des_addr) {
pthread_mutex_lock( &mp_pairs) ;  //  ### lock scans[],
                            scan_pair( src_addr, des_addr, BOOMERANG, 0) ;  // type 8 ->
circle packet or "land" attack
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                            log_pkt = 1 ;
                            goto read_next;
                    }

                    bytes_hdr =  (((unsigned int) ip_buf[2] << 8 ) | (unsigned int)
ip_buf[3]); //includes IP, TCP/UDP/ICMP hdrs
                    bytes = bytes_hdr -m - 8   ;        // length of IP data  (includes
App. Header) UDP, ICMP = 8 bytes
                    if(transport == 6)  bytes += 8 - 4 * (tcp_buf[12] >> 4)  ;  //
adjust for longer TCP header

// Traffic Stats - IP Spoof Detect
                    {
                            unsigned int traf = 0, s, d ;
                            for ( n = 0 ; n < ln_max ; n++) {
                                    if( ! ( local_mask[ n ] & ( local_net[ n ] ^
src_addr ))) traf |= 1 ; // source is local
                                    if( ! ( local_mask[ n ] & ( local_net[ n ] ^
des_addr ))) traf |= 2 ; // destin is local
                }
                            if( ! (0xf0000000 & (0xe0000000 ^ des_addr) ) ) traf |= 4 ;
//  => 4 (outside) or 5 (local) mcast

                            if( (!(des_addr & 0xff000000)) || (((des_addr & 0xff000000))
== 0xff000000 )
                                    || ((des_addr & 0x000000ff)) == 0x000000ff  ) traf
|= 8 ;  // local broadcast   => 9, if 8 then bad
                            if( ! (0xff000000 & (0xff000000 ^ des_addr) ) ) traf |= 8 ;
//  => 4 (outside) or 5 (local)
                            if( ( src_addr & 0xff000000) == 0 ) {
                                    if ( !( traf & 10 )  )     traf |= 16 ; // bad if
dest is not local or local broadcast (traf = 8 or 2)
                                    else traf = 9 ;  // set to local broadcast
                            }
                    switch( traf ) {
                            unsigned int  ips, ipd ;

                            case 0 :  // outside-outside traffic

                                    spoofs++ ;
                                    bytes_oo_cnt +=  ( (unsigned int) ip_buf[2] << 8 ) |
(unsigned int) ip_buf[3] ;
```

```
                                if(spoofs < 10 ) {
                                        if(! f_encrypt) {     //encrypt routine 'c'
                                                printf(" SPOOF PACKET- Source
%3u.%3u.%3u.%3u, Dest. %3u.%3u.%3u.%3u   t: %u\n",
                                                 (src_addr>>24) & 0xff,
(src_addr>>16) & 0xff, (src_addr>>8) & 0xff, src_addr & 0xff,
                                                 (des_addr>>24) & 0xff,
(des_addr>>16) & 0xff, (des_addr>>8) & 0xff, des_addr & 0xff,t_run);
                                        } else {
                                                ips = src_addr ^
(((src_addr>>4)&0x0f0f0000) ^ (src_addr<<4)&0x0000f000) ;
                                                ipd = des_addr ^
(((des_addr>>4)&0x0f0f0000) ^ (des_addr<<4)&0x0000f000) ;
                                                printf(" SPOOF?- Source
%3u.%3u.%3u.%3u, Dest. %3u.%3u.%3u.%3u   t: %u\n",
                                                  (ips>>24) & 0xff, (ips>>16) &
0xff, (ips>>8) & 0xff, ips & 0xff ,
                                                  (ipd>>24) & 0xff, (ipd>>16) &
0xff, (ipd>>8) & 0xff, ipd & 0xff, t_run ) ;
                                        } // end  -  spoofs < 10
                                }
                        break  ;

                        case 1 :  // inside-outside traffic  0001
                                bytes_in_cnt +=  bytes_hdr ;  // bytes including
headers
                        break  ;

                        case 2 :  // outside-inside traffic   0010
                                bytes_out_cnt += bytes_hdr ;
                        break  ;

                        case 3 :  // inside-inside traffic   0011
                                bytes_loc_cnt += bytes_hdr ;
                        break ;

                        case 4 : // outside multicast    0100
                                bytes_mcn_cnt += bytes_hdr ;
                                        if( hs ){
                                                host[ hs ].server |= port_mask[ 2050
] ; // mcast_bit
                                }
                                goto read_next ;
                        break ;

                        case 5 : // inside broadcast     0101
                                bytes_mco_cnt += bytes_hdr ;
                                        if( hs ) {  // create entry if none exists
                                                host[ hs ].server |= port_mask[ 2050
] ;// mcast_bit
                                }
                                goto read_next ;
                        break ;

                        case 9:        // inside broadcast to 0.0.0.0     1001

                        case 11:     // inside broadcast to x.x.x.255     1011
                                bytes_bc_cnt += bytes_hdr ;
                                        if( hs ) {  // create entry if none exists
                                                host[ hs ].server |= port_mask[ 2 *
UDP_PORT_OFFSET + 3 ] ;// bcast_bit
                                }
                                goto read_next ;
                        break ;

                        default :
                                bytes_bad_cnt += bytes_hdr ; // outside to broadcast
        1001
                                        log_pkt = 1 ;
                                goto read_next ;
                        }  //  End case (traf)
          } // end traffic stats

// ================= CHECK FOR PATHOLOGICAL PACKETS =============(LINE
651)============ //


//  TCP and UDP Flow numbers
                if((transport == 6) || (transport == 17)) { // collect flow data

        src_port  = ((unsigned long) tcp_buf[0] << 8) | ((unsigned long) tcp_buf[1]
);// src port
        des_port  = ((unsigned long) tcp_buf[2] << 8) | ((unsigned long) tcp_buf[3]
);// des port

                        if( hs ) {
                                host[ hs ].bytes_ot_pp += bytes_hdr ; // bytes_ot_pp
is bytes out per period (0'ed every 5 min.)
                                host[ hs ].pkts_ot ++ ;

                                if ( (transport == 6 )  && (flag_alr[ tcp_buf[13]
])) { // bad-flags TCP
                                        host[ hs ].bad_pkts ++ ;
                                        log_pkt = 1 ;
                                        record_probe( hs, des_addr, des_port ) ;

                                        if( host[ hs ].bad_pkts < 1000 ) {
pthread_mutex_lock( &mp_pairs) ;    //  ### lock scans[],
                                                scan_pair( src_addr, des_addr,
BAD_PKT_TRACE, des_port) ;
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                        }       }

                        if( transport == 17) {    //  UDP
                                host[ hs ].udp_bytes += bytes_hdr;  // bytes_hdr
includes header bytes
                                host[ hs ].bytes_ot += bytes_hdr ;

                                if (bytes <= SHORT_UDP_MAX ) {  // too short
                                        host[ hs ].bad_pkts ++ ;             //
too-short UDP
                                        host[ hs ].alerts |= SHORT_UDP ;     //
too-short UDP
                                        log_pkt = 1 ;
                                        record_probe( hs, des_addr, des_port ) ;
        pthread_mutex_lock( &mp_pairs) ;    //  ### lock scans[],
                                        scan_pair( src_addr, des_addr,
SHORT_UDP_SCAN, des_port) ;
        pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                                }
                        }
                        else    host[ hs ].bytes_ot += bytes ; // TCP, data bytes
only

                        if(( ip_buf[8] < 2 ) && des_addr) {  //  TTL expiring, TCP,
UDP, OR ICMP, ignore b'cast to 0.0.0.0
                                host[ hs ].traces ++ ;
                                host[ hs ].alerts |= TRACE_ALERT ;  // ICMP timeouts
are ignored
                                log_pkt = 1 ;
                                record_probe( hs, des_addr, src_addr ) ;
pthread_mutex_lock( &mp_pairs) ;    //  ### lock scans[],
                                if (transport ==  6) scan_pair( src_addr, des_addr,
TCP_TO, des_port) ;
                                if (transport == 17) scan_pair( src_addr, des_addr,
UDP_TO, des_port) ;
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                        }
                }

                            //   ============ END - LOOKING FOR PATHOLOGICAL PACKETS
========== //

                if( hd ) {
                        host[ hd ].bytes_in_pp += bytes_hdr ; // bytes_in_pp is
bytes in per period (0'ed every 5 min.)
                        host[ hd ].pkts_in ++ ;
                        if( transport == 17) {   //  UDP
                                host[ hd ].udp_bytes += bytes_hdr; // total UDP, in
and out, includes header bytes
                                host[ hd ].bytes_in += bytes_hdr ;
                        }
                        else    host[ hd ].bytes_in += bytes ; // TCP
                }

                if ( src_addr < des_addr ) {  // ip0 is the smaller IP addr
                        ip0 = src_addr ;        port0 = src_port ;
                        ip1 = des_addr ;  port1 = des_port ;
                        h0 = hs        ;  h1 = hd          ;
                } else  {
                        ip1 = src_addr ;  port1 = src_port ;
                        ip0 = des_addr ;  port0 = des_port ;
                        h0 = hd        ;  h1 = hs          ;
                }

                if( (n_slot = find_slot(ip0, ip1, port0, port1)) == 0) {  // #####
Find slot in Flow data structure

                                printf(" *** Flow Data Full np: %u, Scan_max: %u,
ip: %u, pt: %u, ip1: %u, pt1: %u\n",
                                                 np,scan_max,ip0,port0,ip1,port1) ;
                                goto read_next ;
                        }
#ifdef CHECK_INDEX
                        if(n_slot > SLOTS ) {
                                printf("Bad n_slot: %u, np:%u, t_run: %u\n", n_slot,
np, t_run );
                                fflush(stdout) ;
                                break ;
                        }
#endif
                        if((host[hs].alerts & WATCH_HOST) || (host[hd].alerts &
WATCH_HOST)) {
                                flow[n_slot].state |= WATCH_FLOW ;  // LOG THIS FLOW
AND ALL PACKETS
                        }
                        if((scan_max > SCAN_MAX) && (scan_max> flow[n_slot].scans))
{  // === PORT SCAN DETECTION
```

```
                                flow[n_slot].scans = scan_max ;
                                log_pkt = 1 ;
                        }

                        if( ! (flow[n_slot].state & NOT_FIRST_PKT) ) { //  this is
FIRST PACKET

                                if(( host[ hs ].alerts | host[ hd ].alerts) &
ALARM_12W ){
                                        flow[n_slot].state |= WATCH_FLOW ;  // log
all flows and pkts
                                }

                                if( flow[n_slot].state & LOOSE  == 0 ){ // not LOOSE
and  FIRST PACKET

                                        if ( src_addr < des_addr )      {
                                                flow[n_slot].state  |= ip0_FIRST ;
                                                flow[n_slot].service =
//
flow[n_slot].pt[1] ;
                                        }
                                        else {
                                                flow[n_slot].state |= ip1_FIRST ;
                                                flow[n_slot].service =
//
flow[n_slot].pt[2] ;
                                        }

                                        if( (transport ==  6) && ( tcp_buf[13] !=
0x02 ) && (! f_loose )) {  // not SYN, 1st packet not seen
                                                if(( port0 != 25) && (port1 != 25))
flow[n_slot].state |= NO_SYN ;  // do not set for Telnet

                                        } // end - no SYN
                                } // not LOOSE
                        } // end - 1st packet in flow
                }  // end TCP or UDP

        switch( transport )  // ACTION FOR DIFFERENT TRANSPORT PROTOCOLS
                {
        case 6 :        //  *** TCP ***
                n_tcp++ ;
                flags = (unsigned int) tcp_buf[13] | flag_alr[ tcp_buf[13] ] ;  //
look up  flag combinations, set bits 0,1,2,3

                ah = (tcp_buf[12] >> 2) & 0x3c ;  //  tcp_buf[ah] == 1st application
header byte ;

                if( ip0 == src_addr) {
                        flow[n_slot].pkts[0]++ ;
                        flow[n_slot].bytes[0] += bytes ; // does not include IP hdr
                        flow[n_slot].flgs[0] |= flags ;

                        if( (rec_len - both_hdr - m - 20)  >= 4)  {  // app.header
>= 4 bytes long
                                if( flow[n_slot].bin_norm[0] < 0xffff )
flow[n_slot].bin_norm[0] += 1 ;
                                else {
                                        flow[n_slot].bin_norm[0] = 0xcfff ;
                                        flow[n_slot].binary[0] =
(flow[n_slot].binary[0] >> 1) + ( flow[n_slot].binary[0] >> 2 ) ; // 3/16
                                }
flow[n_slot].binary[0] +=
(tcp_buf[ah]>>7)+(tcp_buf[ah+1]>>7)+(tcp_buf[ah+2]>>7)+(tcp_buf[ah+3]>>7);
```

```
                        }

                        if( flow[n_slot].flag_norm[0] < 0xffff )
flow[n_slot].flag_norm[0] += 1 ; // normalize if needed
                        else {
                                flow[n_slot].flag_norm[0] = 0xcf ;
                                flow[n_slot].flag[0][0] = (flow[n_slot].flag[0][0]
>> 1) + ( flow[n_slot].flag[0][0] >> 2 )
                                        + (flow[n_slot].flag[0][0] & 1 ) ; // keep >
0 if ever > 0
                                flow[n_slot].flag[0][1] = (flow[n_slot].flag[0][1]
>> 1) + ( flow[n_slot].flag[0][1] >> 2 )
                                        + (flow[n_slot].flag[0][1] & 1 ) ;
                                flow[n_slot].flag[0][2] = (flow[n_slot].flag[0][2]
>> 1) + ( flow[n_slot].flag[0][2] >> 2 )
                                        + (flow[n_slot].flag[0][2] & 1 ) ;
                                flow[n_slot].flag[0][3] = (flow[n_slot].flag[0][3]
>> 1) + ( flow[n_slot].flag[0][3] >> 2 )
                                        + (flow[n_slot].flag[0][3] & 1 ) ;
                                flow[n_slot].flag[0][4] = (flow[n_slot].flag[0][4]
>> 1) + ( flow[n_slot].flag[0][4] >> 2 )
                                        + (flow[n_slot].flag[0][4] & 1 ) ;
                                flow[n_slot].flag[0][5] = (flow[n_slot].flag[0][5]
>> 1) + ( flow[n_slot].flag[0][5] >> 2 )
                                        + (flow[n_slot].flag[0][5] & 1 ) ;
                        }
                        //           update flag counters when ip0 is source
                        if( flags & 0xffffffc0 ) {
                                 flow[n_slot].flag[0][0] ++ ; // flags >> 0x3f//
SYN-FIN et al
                        }
                        flow[n_slot].flag[0][1] += (flags >> 2) & 0x01 ;  // RESET
                        flow[n_slot].flag[0][2] += (flags >> 5) & 0x01 ;  // URGENT
                        flow[n_slot].flag[0][3] += ((~flags >> 4) & (flags >> 1) &
0x01) ; // not-ACK & SYN
                        flow[n_slot].flag[0][4] += (( flags >> 4) & (flags >> 1) &
0x01) ; //    ACK & SYN
                        flow[n_slot].flag[0][5] += (( flags >> 4) & flags       &
0x01) ;  //    ACK & FIN
                        flow[n_slot].flag[0][6] |= (tcp_buf[18] | tcp_buf[19]) ; //
URG pointer
                }
                else
                { //  ip1 == src_adr

                        flow[n_slot].pkts[1]++ ;  //  [1] indicates ip[1] sent
                        flow[n_slot].bytes[1] += bytes ;// does not include IP hdr
                        flow[n_slot].flgs[1] |= flags ;

                        if( (rec_len - both_hdr - m - ah)  >= 4)  {
                                if( flow[n_slot].bin_norm[1] < 0xffff )
flow[n_slot].bin_norm[1] += 1 ;
                                else {
                                        flow[n_slot].bin_norm[1] = ( unsigned char)
0xcf ;
                                        flow[n_slot].binary[1] =
(flow[n_slot].binary[1] >> 1) + ( flow[n_slot].binary[1] >> 2 ) ; // 3/16
                                }
flow[n_slot].binary[1] +=
(tcp_buf[ah]>>7)+(tcp_buf[ah+1]>>7)+(tcp_buf[ah+2]>>7)+(tcp_buf[ah+3]>>7); //add 0-4
                        }

                        if( flow[n_slot].flag_norm[1] < 0xffff )
```

```
flow[n_slot].flag_norm[1] += 1 ; // normalize if needed
                        else {
                                flow[n_slot].flag_norm[1] =  0xcf ;
                                flow[n_slot].flag[1][0] = (flow[n_slot].flag[1][0]
>> 1) + ( flow[n_slot].flag[1][0] >> 2 )
                                        + (flow[n_slot].flag[1][0] & 1 ) ; // 3/4
                                flow[n_slot].flag[1][1] = (flow[n_slot].flag[1][1]
>> 1) + ( flow[n_slot].flag[1][1] >> 2 )
                                        + (flow[n_slot].flag[1][1] & 1 ) ;
                                flow[n_slot].flag[1][2] = (flow[n_slot].flag[1][2]
>> 1) + ( flow[n_slot].flag[1][2] >> 2 )
                                        + (flow[n_slot].flag[1][2] & 1 ) ;
                                flow[n_slot].flag[1][3] = (flow[n_slot].flag[1][3]
>> 1) + ( flow[n_slot].flag[1][3] >> 2 )
                                        + (flow[n_slot].flag[1][3] & 1 ) ;
                                flow[n_slot].flag[1][4] = (flow[n_slot].flag[1][4]
>> 1) + ( flow[n_slot].flag[1][4] >> 2 )
                                        + (flow[n_slot].flag[1][4] & 1 ) ;
                                flow[n_slot].flag[1][5] = (flow[n_slot].flag[1][5]
>> 1) + ( flow[n_slot].flag[1][5] >> 2 )
                                        + (flow[n_slot].flag[1][5] & 1 ) ;
                        }
                        //           update flag counters when ip1 is source
                        if( flags & 0xffffffc0 ) {
                                 flow[n_slot].flag[1][0] ++ ; // flags >> 0x3f//
SYN-FIN et al
                        }
                        flow[n_slot].flag[1][1] += (flags >> 2) & 0x01 ;  // RESET
                        flow[n_slot].flag[1][2] += (flags >> 5) & 0x01 ;  // URGENT
                        flow[n_slot].flag[1][3] += ((~flags >> 4) & (flags >> 1) &
0x01) ; // not-ACK & SYN
                        flow[n_slot].flag[1][4] += (( flags >> 4) & (flags >> 1) &
0x01) ; //    ACK & SYN
                        flow[n_slot].flag[1][5] += (( flags >> 4) & flags       &
0x01) ;  //    ACK & FIN
                        flow[n_slot].flag[1][6] |= (tcp_buf[18] | tcp_buf[19]) ; //
URG pointe
                }
                //  ====  APPLICATION LAYER CHECK ====
                app_0 = 4 * (tcp_buf[12] >> 4 ) ;  // TCP options between 20 and
data_0
                app_len = rec_len  - ip_options_1  - both_hdr- app_0; // index in
tcp_buf, rec_len - all hdrs
                if( app_len >= 5 ) {
                        if( ! memcmp( &tcp_buf[ app_0 ], "ISON ", 5 )) log_pkt = 1 ;
                        if( ! memcmp( &tcp_buf[ app_0 ], ":irc.", 5 )) log_pkt = 1 ;

//                                host[hs].server |= IRC ;
//                                host[hd].client |= IRC ;
                }  // end - app_len > 4

        break ; // end TCP

        case 17 :       // *** UDP ***
                {
                int j, k, d ;
//              data_0 = 8   ; // 1st byte of data in tcp_buf
//              data_1 = rec_len  - ip_options_1 - both_hdr ; // last byte of data
in tcp_buf
                d  =  rec_len  - ip_options_1 - both_hdr - 8 ;
                n_udp++ ;
                if( ip0 == src_addr)  j = 0 ; //  index 'j' is source
                else j = 1 ;
        k = 1 - j ;

        flow[n_slot].pkts[j]++ ;
        flow[n_slot].bytes[j] += bytes ; //does not include TCP/IP hdr
        if( d <= SHORT_UDP_MAX ) {
            flow[n_slot].flag[j][BAD] ++ ;
            log_pkt = 1 ;
        }
        if((flow[n_slot].pt[k] == 7) || (flow[n_slot].pt[k] == 19)) { //
echo and chargen
            flow[n_slot].flag[j][BAD] ++ ;
            log_pkt = 1 ;
        }

    }
    break ; // end of UDP

    case 1 :    // *** ICMP ***  - no flows for ICMP
        n_icmp++ ;
//char icmp_type[18][20] = {"Echo_Reply", "Type_1", "Type_2", "Des_Unreach",
"Quench", "Redirect", \
//"Type_6", "Type_7", "Echo_Req", "Rtr_Advert", "Rtr_Solic", "Time_Out",
"Param_Prob", \
//"Timestamp", "Time_Reply", "Info_Request", "Info_Reply", "Type_>16"} ;

        if( hs  > 0 ) {
            host[hs].pkts_ot++ ;
            host[hs].bytes_ot    += bytes     ; // does not include IP
hdr
            host[hs].bytes_ot_pp += bytes_hdr ; // includes IP hdr
            host[hs].pkts_ot ++ ;
            if( tcp_buf[0] == 8 ) {           //  Echo_Req
                record_probe( hs, des_addr, des_port ) ;  // hs is
source of probe, des_addr is destination
                host[hs].pings ++ ;
                if( host[hs].pings > 200 ) {
                    host[hs].alerts |= PING_ALERT ;
pthread_mutex_lock( &mp_pairs) ;  //  ###  lock scans[],
//xx                scan_pair( src_addr, des_addr, PING_SCAN, 0)
;  // ping probe
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
            }       }
        }

        if( hd  > 0 ) {
            host[hd].pkts_in++ ;
            host[hd].bytes_in    += bytes     ; // does not include IP
hdr
            host[hd].bytes_in_pp += bytes_hdr ; // includes IP hdr
            host[hd].pkts_in ++ ;

            if( tcp_buf[0] == 0 ) {    //  Echo_Reply
                host[ hd ].pings ++ ;
                record_probe( hd, src_addr, des_port ) ;  // hd is
source of probe, src_addr is destination
                if( host[hd].pings > 100 ) {
                    host[ hd ].alerts |= PING_ALERT ;
//xx pthread_mutex_lock( &mp_pairs) ;  //  ###  lock scans[],
//xx                scan_pair( des_addr, src_addr, PING_SCAN, 0)
;
//xx pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
            }       }
         if( tcp_buf[0] == 3 )  {
```

```
                                host[ hd ].rejects ++ ;  //  Des_Unreach
                                log_pkt = 1 ;
                                record_probe( hd, src_addr, des_port ) ;  // hd is
source of probe, src_addr is destination
                        }  //  end - des unreachable

                        if( tcp_buf[0] == 11 ) {                              //  Time-Out
Notice
                                register unsigned int d_ip, i_pt ;
                                register unsigned short d_pt ;
                                host[hd].traces ++ ;
                                d_ip =  s2i( &tcp_buf[ 24 ] ) ; // original
destination in ICMP 8 + ip2[16]
                                i_pt = 10 + 4*(tcp_buf[8] & 0x0f) ; // index in
tcp_buf to original src port number
                                d_pt = 256 * tcp_buf[ i_pt ] + tcp_buf[ i_pt + 1] ;
                                record_probe( hd, d_ip, d_pt ) ;
                                if( host[hd].traces > 100 ) {
pthread_mutex_lock( &mp_pairs) ;  //  ###  lock scans[],
                                        scan_pair( des_addr, d_ip, ICMP_TO, 0) ; //
20 + length of IP header
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                        }       }
                }       //  end hd > 0

//              data_0 = 8 ;
//              data_1 = rec_len - ip_options_1 - both_hdr ; // last byte of data in
tcp_buf
        break ; // end of ICMP

        default:        not_tui++ ;  // count packets not TCP, UDP, or ICMP
(discarded)
        }  //  end of switch on ip_buf[9] - layer 4 protocol


        // ==============================  READ NEXT PACKET RECORD
================================
read_next:
                                          ;  // log all flows and pkts

                if(f_demo || log_pkt || flow[n_slot].state & WATCH_FLOW ) {
                        fwrite(  snp_buf, sizeof(char), 24      , log_pkt_file) ; //
 write tcpdump packet header
                        fwrite(  lan_buf, sizeof(char), lan_hdr , log_pkt_file) ; //
 write tcpdump LAN  header
                        fwrite(  ip_buf , sizeof(char), tcp_all , log_pkt_file) ; //
 write tcpdump IP & TCP  header
                }
if( f_verbose == 3 ) { printf("  Pkt %u done at %u\n", np, t_run ) ; fflush(stdout)
; }
pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
pthread_mutex_unlock( &mp_flows) ;  //  ### unlock flows[]



                if(f_file)  {  //       --------------------timing control for FILE
INPUT
                        if( t_run > t_pause ) {
                                if( f_verbose == 2)  printf(".") ; // should be 30
.'s then \n

                                t_pause += 3 ;
                        }
                        if((t_run + t_zero) > t_next_cf) {
                                t_next_cf += CLASS_FLOW_INT ;
                                if( f_verbose == 2)  printf("\n") ;
                                fflush( stdout ) ;
                                pthread_mutex_unlock( &mp_class_flow ) ; // clear
and continue
                                if( f_verbose == 2)  printf( "  ## process_pkts -
blocking for other threads. ##\n" ) ;  fflush( stdout ) ;
                                sleep(2) ;  // give class_flows() a chance to run
                                pthread_mutex_lock(   &mp_class_flow ) ; // block
until class_flow() is done
                        }
                } // end f_file true

                n = fread( (void *) snp_buf, 4, 6, infile) ;            // read
snoop record header into snp_buf[ ]
                if( n<6) { printf("\nEOF at Packet %u\n", np) ; break ; }
read_ip_buf:

        if( snoop ) { // snoop
                rec_len = ((unsigned long) snp_buf[8] << 24) | ((unsigned long)
snp_buf[9] << 16) | \
                                        ((unsigned long) snp_buf[10] << 8) |
(unsigned long) snp_buf[11] ;
                t_read = ((unsigned long) snp_buf[16]<<24) | ((unsigned long)
snp_buf[17] << 16) | \
                               ((unsigned long) snp_buf[18]<<8) | (unsigned long)
snp_buf[19] ;
        } else {   // tcpdump
                rec_len = ((unsigned long) snp_buf[11] << 24) | ((unsigned long)
snp_buf[10] << 16) | \
                                        ((unsigned long) snp_buf[9] << 8)  |
(unsigned long) snp_buf[8] + 24 ;

                                                                   // tcpdump does not
include it's 24
                t_read = ((unsigned long) snp_buf[3] << 24)  | ((unsigned long)
snp_buf[2] << 16) | \
                                        ((unsigned long) snp_buf[1] << 8) |
(unsigned long) snp_buf[0] ;
        }

                if( rec_len > snp_rec_max ) {
                        printf("\n*** FILE OUT OF SYNC *** after %u, rec_len %u, not
%u\n\n", np, rec_len, snp_rec_max);
                        break ;
                }
                n = fread( (void *) lan_buf, 1, lan_hdr , infile); // read IP+TCP or
UDP header into lan_buf[ ]
                if ( n < lan_hdr )  {
                        printf("EOF after Packet %u", np) ;
                        break ;
                }
                tcp_all = rec_len - both_hdr ;
                n = fread( (void *) ip_buf, 1, tcp_all , infile) ; // read IP+TCP
header into ip_buf[ ]
                if ( n < tcp_all )  {
                        printf("EOF after Packet %u\n", np) ;
                        break ;
                }
                np++ ;
```

```
//if( (np % 100000) == 0 ) { printf(".\008") ; fflush(stdout) ; } // progress dots
//      t_check = time(NULL) ;
//  printf("np: %u, n_flow: %u, active: %u, max: %u, t_run: %u, Real t: %u\n",
//                      np, n_flow, n_active, n_active_max, t_run, (t_check -
t_now)) ; ///DEBUG
//  fflush(stdout) ;

        }       // #################### break -> END OF LOOP TO READ PACKETS
#########################

        if( (f_file == 0 ) && (running == 1 ) ) {    // === TRY TO RESTART TCPDUMP
printf( "\n ### TCPDUMP STOPPED SENDING DATA, will try to restart. np: %u, t_run: %u
sec\n\n",np,t_run);
        fclose( infile ) ;
                for( i = 0 ; i < 3600 ; i++ ) {
                        if( running != 1 ) break ;
                        if( start_tcpdump( ) == 0 ) {  // returns '0' if tcpdump
restarted successfully
                                n = fread( (void *) snp_hdr, 4, 6, infile) ;    //
read new file header
                                goto restart_here ;
                        }
                        else sleep( 24 ) ;
                fclose( infile ) ;
                }               // end if f_file == 0   -> realtime
        }
        running = 2 ; // RESTART NOT SUCESSFUL, or f_file,  => close all threads

//      if( f_file) pthread_mutex_unlock( &mp_class_flow ) ; // clear and let
process_pkts continue
printf("Thread process_pkts ending at t: %u, np: %u\n", t_run, np) ;
        return ;
} //  ### end process_pkts ###


int start_tcpdump( void ) // ===================   START (and RESTART) TCPDUMP
==================
{
                FILE * tmpfile ;
                register int i, j ;
                char c, command[200], select[120] ;
                if( setuid(0) != 0 ) {
                        printf( "  ###  CAN NOT SETUID(0) and START tcpdump -
TERMINATING ###\n") ;
                        return( 1 ) ;
                }
                sprintf(command,"date\n");
                printf("   system commands: %s", command) ;         // this will
print date and time
                system(command);

                kill_tcpdump() ;  // kill any present instance of tcpdump
                sleep(1) ;
                sprintf(command,"/usr/bin/mkfifo myfifo");
                printf(" : %s\n", command);         // this will create a new
"myfifo"
                if( system(command) ) printf(" #### FAILED ####");  // reset
interface eth1

                sprintf(command,"/sbin/ifconfig eth1 down promisc up");
                printf(" : %s\n", command) ;         // reset eth1 to promiscuous
and start it up
                if( system(command) ) printf(" #### FAILED ####");  // reset
```

```
interface eth1

sprintf(command,"/bin/nice -20 /usr/sbin/tcpdump -lni eth1 -s 120 -w myfifo 'ip'
1>>log_tcpdump 2>>log_tcpdump &");
                printf(" : %s\n", command) ;         // kill( (pid_t) k, SIGQUIT ) ;
                sleep(1) ;
                if( 0 == system(command)) {   // start 'tcpdump' on interface eth1
                        if((infile = fopen ("myfifo","rb"))!=NULL) {   //  OPEN
INPUT FILE  //  File opening for input from TCPDUMP
                                printf("  tcpdump STARTED SUCESSFULLY.\n") ;
                                return( 0 ) ;
                        }
                }
                printf("  tcpdump START or 'myfifo' OPEN - FAILED.\n") ;
                return( 1 ) ;
} // end start_tcpdump


void kill_tcpdump( void ) {
        int i, j, k ;
        char command[100], select[100], c ;
        FILE *tmpfile ;

                system("/bin/ps -e | /bin/grep 'tcpdump' >      tempfkiebtj" ) ;  //
see if TCPDUMP running

                if( (tmpfile = fopen("tempfkiebtj","r")) != NULL) {

                        for( i = 0 ; i < 99 ; i++ ) {
                                if( ( (c = getc(tmpfile)) != EOF) && ( c >= 32) ) {
                                        select[i] = c ;
                    }
                                else {
                                        select[i] = 0 ;
                                        break ;
                        }       }
                        fclose(tmpfile) ;
                        system("/bin/rm -f tempfkiebtj") ;

                        while( (select[i] <= ' ') && (i > 0) ) { // remove spaces at
end, if any
                                select[i] = 0 ; i-- ;
                        }

                        if( (i>7) && ( 0 == strcmp("tcpdump", &select[i-6])) && (
select[i-7] == ' ') ) {

                                j = -1 ;
                                for( i = 0 ; i < 99 ; i++ ){
                                        if((select[i] == ' ') && (j >=0)) {
                                                select[i] = 0 ;
                                                break ;
                                        }
                                        if((select[i] != ' ') && (j < 0)) j = i ;
                                }
                                k = atoi(&select[j]) ;
                                if( (k > 0 ) && ( k <= 99999) ) {
                                        sprintf(command,"/bin/kill %d", k);
                                        printf("   system command sent to kill
tcpdump: %s ", command) ;  // kill( (pid_t) k, SIGQUIT ) ;
                                        if( system(command) ) printf(" #### FAILED
####\n");  //  kill tcpdump, if necessary
                                        else printf(" --- SUCCESS ---\n");
                                        sleep(1) ;
                                }
                        }
                } //  if file tempfkiebtj opened - tcpdump now "killed" if it was
still running


                sprintf(command,"/bin/rm -f myfifo");
                printf(" : %s  ", command) ;         // this will remove "myfifo"
(and junk still in it)
                if( system(command) ) printf(" #### FAILED ####\n");   // reset
interface eth1
                else printf(" --- SUCCESS ---\n");
}   // end - kill_tcpdump()


void usage( void ) {
        printf("\nUsage: lancope   in-file    out-file  flags\n");
        printf("   Output file extension should be .csv for StarOffice, .txt for
Excel\n");
        printf("   Output file  = '-' means date-derived file, yymmddhh.txt , e.g.
'00081515.txt\n" );
        printf("   Flags: d = dotted decimal, e = encrypt local IPs, a = write all
flows\n");
        printf("          f = input from file, s = split_path, o = demo, all on
alert & traffic\n");
        printf("          n = no-read old profiles\n");
        printf("   A file, 'lancope_config.txt' must be in the default directory
with config info\n");
        exit(-1) ;
}


unsigned long   s2i(unsigned char bb[4])  //msB first string to uns long int
   {
       unsigned long j ;
       j = ((unsigned long) bb[0]) << 24 | ((unsigned long) bb[1]) << 16 | \
           ((unsigned long) bb[2]) <<  8 | ((unsigned long) bb[3]) ;
       return(j);
   } //end s2i()

unsigned long   b2i(unsigned char bb[4])  //lsB first string to uns long int
   {
       unsigned long j ;
       j = ((unsigned long) bb[3]) << 24 | ((unsigned long) bb[2]) << 16 | \
           ((unsigned long) bb[1]) <<  8 | ((unsigned long) bb[0]) ;
       return(j);
   } //end b2i()



long find_slot( unsigned long ip0, unsigned long ip1, long port0, long port1)
      //  returns 0 if no slot available, i if entry exists // mp_flows LOCKED
{
        unsigned long x, i, i_mark, n, index, h, flows_max = SLOTS, unlock = 0 ;
        int hihi = 0 , ftp_server = 0 ;
        if(! pthread_mutex_trylock( &mp_flows ) ) {
                        printf(" ### find_slot() without mutex m_flow being set. t:
%u, Running: %u\n",t_run,running);
                        unlock = 1 ;
        }
        x =  ip0 ^ (ip1<<11)^(ip1>>21) ;  //***  make index from DA, SA, check Port
***/
        index = x ;
        for (i = SHIFT ; i < 32 ; i += SHIFT ) {
                x = x >> SHIFT ;
                index ^=  x ;           //  XOR bytes together
        }
        index  = index & MASK ;  // limit to right SHIFT bits
        index++ ;               // index can be 1 to 2^n +1

                //  --- look for matching connection or first empty slot
        scan_max = 0 ;  // count max ports connected
        if(( port0 > 1023) && (port1 > 1023))  hihi = 1 ;
        if(( port0 == 21) || ( port0 == 20 ))  ftp_server = 1 ;
        if(( port1 == 21) || ( port1 == 20 ))  ftp_server = 2 ;

        if( flow[index].root) {    //   root not empty, run through list, look for
matching flow
                i = flow[index].root ;
                do {
                        register int nomatch, type ;
                        nomatch = 1 ; // test match below
                        if( (flow[i].ip[0] == ip0) && (flow[i].ip[1] == ip1) ) { //
nomatch if IP's different
                                scan_max++ ;                       //max ports open
ip0-ip1
                                type = ( flow[i].state >> 1 ) & 3 ; // type = 0, 1,
2

                        if((flow[i].service & HI_HI)&&(!(flow[i].service &
FTP_ANY))) {
                                if( ftp_server == 1 ) flow[i].state |= FTP_SER_0 ;
// new flow is marked for FTP_0/1
                                if( ftp_server == 2 ) flow[i].state |= FTP_SER_1 ;
// and the old one has high-high ports
                        }

                        if(( hihi ) && (! ftp_server)) {

                                if(flow[i].state & FTP_SER_0) ftp_server = 1 ; //
old flow is marked for FTP_0/1 and
                                if(flow[i].state & FTP_SER_1) ftp_server = 2 ; //
the new one has high-high ports
                         } // end - if FTP check

                                switch( type ) {
                                case 0 :
                                        if( (flow[i].pt[0] == port0) &&
(flow[i].pt[1] == port1) ) {  // exact port match
                                                nomatch = 0 ;
                                                break ;
                                        }
                                        if( flow[i].pt[1] == port1) {
                                                nomatch = 0 ;
                                                flow[i].state |= 0x2 ; // type => 1
  ip1 is server, ip0 is client
                                                flow[i].pt_min = flow[i].pt_max =
flow[i].pt[0];
                                                if( port0 < flow[i].pt_min )
flow[i].pt_min = port0 ;
                                                if( port0 > flow[i].pt_max )
flow[i].pt_max = port0 ;
                                                break ;
                                        }
                                        if( flow[i].pt[0] == port0) {
                                                nomatch = 0 ;
                                                flow[i].state |= 0x4 ; // type => 2
 ip0 is Server, ip1 is client
                                                flow[i].pt_min = flow[i].pt_max =
flow[i].pt[1];
                                                if( port1 < flow[i].pt_min )
flow[i].pt_min = port1 ;
                                                if( port1 > flow[i].pt_max )
flow[i].pt_max = port1 ;
                                                break ;
                                        }
                                        break ;
                                case 1 :  // ip0 is client
                                        if( flow[i].pt[1] == port1) {
                                                nomatch = 0 ;
                                                if( port0 < flow[i].pt_min )
flow[i].pt_min = port0 ;
                                                if( port0 > flow[i].pt_max )
flow[i].pt_max = port0 ;
                                                break ;
                                        }
                                        break ;
                                case 2 : // ip1 is client
                                        if( flow[i].pt[0] == port0) {
                                                nomatch = 0 ;
                                                if( port1 < flow[i].pt_min )
flow[i].pt_min = port1 ;
                                                if( port1 > flow[i].pt_max )
flow[i].pt_max = port1 ;
                                                break ;
                                        }
                                        break ;

                                } // end switch( type )
                        }

                        if( nomatch ) {
                                if( (flow[i].up)) {
                                        i = flow[i].up;
                                } else {
                                        break ;   // break out of while(1) - end of
list, no match
                                }


                        } else {  // there was a match
                                goto update_flow ; // slot for flow already exists
                        }
                } while( 1)   ; //  break out (goto) when end of list or match found


                i_mark = i ;      // index of last entry in list - NEW HOST ENTRY

                for( n = 0 ; n < RANGE ; n++) {
                        i += 5 + n ;
                        if( i >= SLOTS ) i = i - SLOTS + 1;   // can not use i = 0
                        if(flow[i].down == 0 ) {  //  empty, use for new entry
                                flow[i_mark].up = i ;
                                flow[i].down = i_mark ;
                                goto new_flow ;
                        }
```

```
            }
            n_full++ ;
            if( unlock ) pthread_mutex_unlock( &mp_flows) ;  //  ### unlock
flows[]
            return( 0 ) ;   // flow data sub-structure full
    }

    else {    //  flow[index].root == 0,  search for any empty slot for NEW
ENTRY, 1st for this ip0-ip1
            i = index ;
            for( n = 0 ; n < RANGE ; n++) {
                    i += 5 + n ;
                    if( i >= SLOTS ) i = i - SLOTS + 1;   // can not use i = 0
                    if(flow[i].down == 0 ) {
                            flow[index].root = i ;
                            flow[i].down = index ;
                            goto new_flow;  //   new leaf
                    }
            }
            n_full++ ;
            if( unlock ) pthread_mutex_unlock( &mp_flows) ;  //  ### unlock
flows[]
            return( 0 ) ;   // flow data sub-structure full
    }

new_flow:

    n_flow++ ;
    n_active++ ;
    if( n > n_max) n_max = n ;  //  check hash efficiency
    flow[i].up    = 0    ;
    flow[i].ip[0]  = ip0  ;
    flow[i].ip[1]  = ip1  ;
    flow[i].pt[0] = port0 ;
    flow[i].pt[1] = port1 ;
    flow[i].start = t_run ;
    flow[i].state = transport & 0x1 ; // 0 -> TCP, 1 -> UDP, Type = 0
    if( f_loose) flow[i].state |= LOOSE ;

    if( ftp_server ) {
            flow[i].service = 21 ;
            if( ftp_server == 1) flow[i].state |= FTP_SER_0  ;  // set bit
            if( ftp_server == 2) flow[i].state |= FTP_SER_1  ;  // set bit
    }
    if( hihi) flow[i].state |= HI_HI  ;  //
update_flow:
    flow[i].last  = t_run ;
    if( unlock ) pthread_mutex_unlock( &mp_flows) ;  //  ### unlock flows[]
    return( i ) ;
}                // end  find_slot()

void  class_flow( void ) { //    #######  CLASSIFY FLOW ###locks flows, then hosts
####
//    ==========
    unsigned int i, j, k, n, s1 = 0, udp, error_count = 0, alarms0 = 0, h0 = 0 ;
    unsigned long h ;
    unsigned int pps0, pps1, flows_ended, n_fprint, mm_bit, mcast_bit, odd_bit,
bcast_bit, wai ;
    char reason[20] ;
    time_t  t_here, t_next_prof ;
    int     this_hour, last_hour, clear ;

t_wa_last = t_zero ;
```

```
mm_bit  = port_mask[   2 * UDP_PORT_OFFSET +1 ] ; //  bit for multimedia ,
psedo-port = 2049
mcast_bit = port_mask[ 2 * UDP_PORT_OFFSET +2 ] ; //  bit for multicast service,
psedo-port = 2050
bcast_bit = port_mask[ 2 * UDP_PORT_OFFSET +3 ] ; //  bit for broadcast service,
psedo-port = 2051
odd_bit = port_mask[   2 * UDP_PORT_OFFSET +4 ] ; //  bit for odd service,
psedo-port = 2052
printf("Thread class_flow start  at t: %u, np: %u, flows: %u\n",
                        (unsigned int) *(&t_run), (unsigned int) *(&np) , (unsigned
int) *(&n_flow) ) ;
if( *(&running) == 1 ) {
        {
                unsigned int cfi = CLASS_FLOW_INT ;  // cfi = 30 s
                unsigned int wai = WEB_ALERT_INT  ;  // wfi = 900
                t_next_cf = *(&t_zero) + 0.5 * cfi ;
                t_next_cf = t_next_cf + cfi - (t_next_cf % cfi) ; // moves to
nearest even value

                t_next_web = t_zero + 1.5 * wai ;
                t_next_web -= ((t_next_web % wai) + 2) ; // moves to nearest even
value * wfi
                t_next_prof = t_next_web + 0.5 * wai - 2 ;  // save profiles between
web_alert calls
//              t_bs_last = t_zero  ; // moves to nearest even value *
TRAF_TABLE_MIN
        }
}
// printf("   class_flow() - t_run: %u, t_here: %u, t_next_cf: %u, t_next_web:
%u\n",
//          *(&t_run), *(&t_run) + t_zero , t_next_cf, t_next_web);
// fflush( stdout ) ;

  last_hour = 100 ;  // will be initial set later - hour of the day, 1 - 23

while( running) {  //  'running' == 1: loop, 'running' == 2: execute once and return
        register unsigned int host0 , alarm0, age ;

 if( f_file) {
                sleep(2) ;
                if( f_verbose == 2)  printf("Class_Flow approching mutext lock.
t_run: %u\n", *(&t_run) ) ;
                fflush( stdout ) ;
                pthread_mutex_lock( &mp_class_flow) ;  //  ###  block process_pkts()

                if( f_verbose == 2)  printf("Class_Flow passed mutext lock.
t_run: %u\n", *(&t_run) ) ;
                fflush( stdout ) ;
        }

  if( f_file) t_here = *(&t_run) + t_zero ;  // t_run always starts at 1
  else  t_here = time( NULL ) ;       // if realtime, use real time

        while((t_here < t_next_cf) && (running == 1) && (! f_file) ) {
                sleep(1) ;  //  stall if running == 1,  then check 'running' and
(t_here < t_next_cf)
          if( f_file) t_here = *( &t_run ) + t_zero ;
          else t_here = time( NULL ) ;
          if( t_here >= t_next_cf) break ;
        }
  if( f_file == 0)  t_run  = t_here - t_zero ;

        if( running == 0 ) break ;
        t_next_cf += CLASS_FLOW_INT ;
        if(f_verbose) {
                printf("* Thread class_flow operating at t: %u, next: %u, np: %u,
flows: %u\n",
            t_here - t_zero, t_next_cf - t_zero, (unsigned int) *(&np), (unsigned
int) *(&n_flow)) ;
        if(f_verbose > 1 )      fflush( stdout ) ;
        }
        n_fprint = flows_ended = 0 ;

        for( i = 1 ; i < SLOTS ; i++ ) {
                unsigned long quiet, p, h[2]  ;
                int s[2], c[2], list ;


// S/C determined by SYN's and SYN-ACK's

                if(flow[i].down) { // flows that exist

pthread_mutex_lock( &mp_flows) ;  //  ###  lock flows[]

                if(flow[i].down) { // confirm after locking

        reason[0] = 0 ;
        clear = 0  ;
        quiet = 70 ; // udp and tcp single die if quiet > 20 sec
        age   = 30 ;
        list   = 0 ;
        if( flow[i].state & UDP_FLOW) udp = UDP_PORT_OFFSET ; // index offset for
port_name
        else udp = 0 ;

        if( (!(flow[i].state & UDP_FLOW)) && (flow[i].pkts[0] >
flow[i].flag[1][RST]) &&
            (flow[i].pkts[1] > flow[i].flag[0][RST]) )  quiet = FLOW_DEAD ;
           //  if TCP and packets in + packets out > resets returned  wait FLOW_DEAD
(300 s)


        if( ( t_run  > ( flow[i].last + quiet ) ) || (running==2) ) { // if flow has
been silent for 'quiet' seconds
                flows_ended++ ;
                clear = 1 ;
        }       // end - check to 'clear'


// =============  CHECK FOR PROBES AFTER 'AGE' ===================//

if(( ! (flow[i].state & ATTACK_CHK ) ) && (t_run > (flow[i].start + age)) ){
                register long int sa = 0, da = 0 ;  // If flow NOT classified, and
'old', check for probes
                unsigned char  probe_type = 0 ;
                unsigned short probe_port = 0 ;
pthread_mutex_lock( &mp_hosts) ;

        flow[i].state |= ATTACK_CHK ; // set so we check once only

        for( j = 0 ; j <= 1 ; j++ ) {
                k = 1 - j ;
                probe_type = 0 ;
                h[j] = find_host( flow[i].ip[j], flow[i].bytes[j] ) ; // h[j] ->
ip[j],  may be '0' if slot does not exist
```

```
                if(!(flow[i].state & UDP_FLOW)){  //  TCP - check for probes
                        if( flow[i].pkts[j] && ( (flow[i].pkts[k] ==
flow[i].flag[k][RST] ) ||
                                (flow[i].pkts[j] == flow[i].flag[j][BAD]) ||
(flow[i].pkts[k] == 0 )) ) {
                                probe_type = TCP_PROBE ;
                }       }// end - TCP probe check

                else if( flow[i].flag[j][BAD] + flow[i].flag[k][RST] ) {  // UDP
Check RST means ICMP port unavailable
                        probe_type = UDP_PROBE ;
                } // end - UDP probe check

        if( probe_type && h[j]) {
                        list = 1 ; strcat( reason, "Probe ") ;
                        record_probe( h[j], flow[i].ip[k], flow[i].pt[k]); // h[j]
is source index of probe, 'k' is destination ip
pthread_mutex_lock( &mp_pairs) ;  //  ### lock scans[],
                        scan_pair( flow[i].ip[j], flow[i].ip[k], probe_type,
flow[i].pt[k] ) ; // ip[j] hit ip[k]:pt[k]
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                flow[i].state |= ( PROBE | CLASSIFIED ) ;  // flow has been
classified as "probe"
                } // end - probe_type > 0
        } // end - j = 0,1
        // ========================= CHECK FOR ATTACKS after 'of age'

        if((!udp) && (!(flow[i].state & ATK_PROBE))){  // if TCP - check for
half-open attack
                for( j = 0 ; j <= 1 ; j++ ) {
                        k = 1 - j ;

                        if(( flow[i].pkts[j] > 16 ) && ( flow[i].flag[j][SYN] >
(0.75 * flow[i].pkts[j] )) &&
                                (flow[i].flag[k][S_A] > 4) ) {  // if half-open
attack, 'j' attacking 'k'
                                list = 1 ; strcat( reason, "HO_Attack ") ;  // will
list only if clear is set for this cycle
                                if( h[j] ) {
                                        record_probe( h[j], flow[i].ip[k],
flow[i].pt[k]);// h[j] is source of probe, des_addr is destination ip
                                        host[ h[j] ].alerts  |= HO_ATTACK ;
                                        host[ h[j] ].concern += HO_ATTACK_INIT_CI ;
                                }  // end - find_host() ok
pthread_mutex_lock( &mp_pairs) ;  //  ### lock scans[],
                                scan_pair( flow[i].ip[j], flow[i].ip[k], HALF_OPEN,
flow[i].pt[k]) ;  // type 8 -> circle packet
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                                flow[i].state |= ( CLASSIFIED | ATTACK ) ;  // flow
has been classified as "attack"
                        } // end - Half-Open Attack found
                } // end - for j = 0,1 and k = 1,0
        } // end - check for half-open (lots of SYN's evoking SYN-ACK's)
pthread_mutex_unlock( &mp_hosts) ;
} // end - #### check for probes & attacks after 'age' ####


                                // ==================== END-OF-LIFE FLOW
CLASSIFICATION ==================== //

if( (!(flow[i].state & CLASSIFIED))  && clear ) { // ### CLASSIFY FLOW, - DETERMINE
SERVER/CLIENT ###
        c[0] = c[1] = s[0] = s[1] = 0 ;
```

```
pthread_mutex_lock( &mp_hosts) ;
        h[0] = find_host( flow[i].ip[0], flow[i].bytes[0] ) ; // h[j] -> ip[j],  may
be '0' if slot does not exist
        h[1] = find_host( flow[i].ip[1], flow[i].bytes[1] ) ;

if(f_verbose==4)                {
        printf(" --  Flow Ended. t_run: %u,  Flow[%u].last: %u, quiet: %u,
t[NOW]-t_zero: %u\n",
                     t_run, i, flow[i].last, quiet, t_here - t_zero ) ;
 printf(" -- Type:%u, FTP: %x, Port0: %u, Port1: %u, Serv: %u, Pmin: %u, Pmax: %u,
syn0: %u, sa0: %u, syn1: %u, sa1: %u\n",
        (flow[i].state & 6) >> 1, (flow[i].state & FTP_ANY) >> 12,
flow[i].pt[0],flow[i].pt[1],flow[i].service,flow[i].pt_min,

flow[i].pt_max,flow[i].flag[0][SYN],flow[i].flag[0][S_A],flow[i].flag[1][SYN],flow[i
].flag[1][S_A] ) ;
}


for( j = 0 ; j <= 1 ; j++ ) {  //  ##### TCP -  Check ip[j] for Port Probing based
on flow data for LOOSE flow #####
        k = 1 - j ;
        if( h[j])  {
                if(flow[i].state & LOOSE) {  // LOOSE  ===   Handle case where not
all packets have been seen
                        if(flow[i].scans >= SCAN_MAX) {
                                if(flow[i].flag[k][RST] >= (flow[i].pkts[k] /2 + 2))
{// if the other guy sent many resets
                                        host[h[j]].pt_scans++ ;
                                        list = 1 ; strcat( reason, "Pt_Scan2") ;
                                        flow[i].state |=  PROBE ;
                                        record_probe( h[j],
flow[i].ip[k],flow[i].pt[k] );  // h[j] is source index of probe, des_addr is
destination ip
        pthread_mutex_lock( &mp_pairs) ;  //  ###   lock scans[],
                                        scan_pair( flow[i].ip[j], flow[i].ip[k],
TCP_PORT_SCAN, 0 ) ; // ip[j] hit multi-ports
        pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                } }     }
                else { // not LOOSE
                        if((flow[i].scans >= SCAN_MAX) || (flow[i].flag[k][RST] ==
flow[i].pkts[k])){ // >SCANMAX or all RSTs
                                if(flow[i].flag[k][RST] >= (flow[i].pkts[k] /2 + 1))
{// if the other guy sent more resets
                                        host[h[j]].pt_scans++ ;
                                        list = 1 ; strcat( reason, "Pt_Scan2 ") ;
                                        flow[i].state |=  PROBE  ;
                                        record_probe( h[j], flow[i].ip[k],
flow[i].pt[k]); // h[j] is source index of probe, 'k' is destination ip
        pthread_mutex_lock( &mp_pairs) ;  //  ### lock scans[],
                                        scan_pair( flow[i].ip[j], flow[i].ip[k],
TCP_PORT_SCAN, 0 ) ; // ip[j] hit ip[k]:pt[k]
        pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                        } }
                }               // end - not LOOSE
        } // end - h[j]
} // end -  j = 0,1    k= 1,0 ----  TCP check for Port Probes, not ATK_PROBE


if( flow[i].state & FTP_ANY ) { //  ===== FTP ========== //
        if( flow[i].state & FTP_SER_0)                          s[0] = c[1] = 1 ;
        else if( flow[i].state & FTP_SER_1)                     s[1] = c[0] = 1 ;
} // end FTP
```

```
else { // not FTP
        for( j = 0 ; j <= 1 ; j++ ) {
                k = 1 - j ;
                if( (!flow[i].flag[j][SYN]) && (!flow[i].flag[k][S_A])
                     && (flow[i].flag[j][S_A] || flow[i].flag[k][SYN])) {
                        s[j] = 1 ;  c[k] = 1 ;                            // ip[j]
is Server,  ip[k] is Client based on flags
                } //flag-3 is SYN, flag-4 is SYN-ACK (S_A)
                if( ! (flow[i].state & 0x6)) {  // type 0  - both ports constant
                        if( c[j] == 1 ) flow[i].pt_min = flow[i].pt_max =
flow[i].pt[j] ; //  ip0 is client
                }
        }// end - j = 0,1  k=1,0

        // S/C determinded by flow Type for UDP and TCP (if not FTP)
        if( flow[i].state & 0x2 ) c[0] = s[1] = 1 ;  // type 1, ip0 is client.
Disagree with flags -> port scan
        if( flow[i].state & 0x4 ) c[1] = s[0] = 1 ;  // type 2, ip1 is client

        if( (! udp) && (flow[i].state & LOOSE) && (! c[0]) && (! s[0]) ) {  // loose
-> syn and s-a may be lost
                if( (flow[i].pt[0] < LOW_PT_MAX ) && (flow[i].pt[1] >= LOW_PT_MAX ))
{  s[0] = c[1] = 1 ; }
                if( (flow[i].pt[1] < LOW_PT_MAX) && (flow[i].pt[0]  >= LOW_PT_MAX ))
{  s[1] = c[0] = 1 ; }
                if((flow[i].pt[0]==22)||(flow[i].pt[0]==37)||(flow[i].pt[0]==53))
s[0] = c[1] = 1 ;
                if((flow[i].pt[1]==22)||(flow[i].pt[1]==37)||(flow[i].pt[1]==53))
s[1] = c[0] = 1 ;// ssh (22) - low client numbers
                if((flow[i].pt[0]==512)||(flow[i].pt[0]==513)||(flow[i].pt[0]==514))
s[0] = c[1] = 1 ; // login, sh, prt also
                if((flow[i].pt[1]==512)||(flow[i].pt[1]==513)||(flow[i].pt[1]==514))
s[1] = c[0] = 1 ;
        }

        if(f_verbose==4)                {
                printf(" -- Before.  s0: %u, c0: %u, s1: %u, c1 :%u", s[0], c[0]
,s[1], c[1] ) ;
        }

        //  --------------- Last_Resort to Determine who (0 or 1) is the Server
        if(( !c[0]) && (!s[0])) {
                for( j = 0 ; j < 2 ; j++ ){  // if one port is a common server port,
assume it is the server port
                        k = 1 - j ;
                        if(( flow[i].pt[j] < UDP_PORT_OFFSET ) && ( port_mask[
flow[i].pt[j] ] )) { // if either port is a common local server port
                                s[j] = c[k] = 1 ;
                        }
                }
        }
} // end - find c[] and s[] when not FTP

        if( c[0] && ( flow[i].pt_min < flow[i].pt[0]))  flow[i].pt_min =
flow[i].pt[0] ;// TELLS IF CLIENT IS 'LOW'
        if( c[1] && ( flow[i].pt_min < flow[i].pt[1]))  flow[i].pt_min =
flow[i].pt[1] ;
        if( s[0] && ( ! flow[i].service) )     flow[i].service = flow[i].pt[0] ; //
TELLS IF SERVER IS 'LOW' OR 'HIGH'
        if( s[1] && ( ! flow[i].service) )     flow[i].service = flow[i].pt[1] ;

if(f_verbose==4)        printf("  After.  s0=%u, c0=%u, s1=%u, c1=%u, Service: %u,
Port_min: %u\n", s[0], c[0] ,s[1], c[1],
                flow[i].service, flow[i].pt_min) ;

// =================== END OF SECTION THAT DETERMINES WHO IS HOST, WHO IS SERVER
============//

host0 = alarms0 = 0 ;

for( j = 0 ; j < 2 ; j++ ){
        k = 1 - j ;  // j,k =  0,1 and 1,0   ==========   'j' is this host, 'k' is
other host

        if( ! (flow[i].state & ATK_PROBE )) {  // if not already known to be a probe
from early check

  //  ##### TCP -  Check ip[j] for Port Probing based on flow data for LOOSE flow
#####

                if(flow[i].state & LOOSE) {  // LOOSE  ===   Handle case where not
all packets have been seen
                        if(flow[i].scans >= SCAN_MAX) {
                                if(flow[i].flag[k][RST] >= (flow[i].pkts[k] /2 + 2))
{// if the other guy sent many resets
                                        host[h[j]].pt_scans++ ;
                                        list = 1 ; strcat( reason, "TCP_Pt_Scan ") ;
                                        flow[i].state |=  PROBE ;
                                        record_probe( h[j],
flow[i].ip[k],flow[i].pt[k] );  // h[j] is source index of probe, des_addr is
destination ip
        pthread_mutex_lock( &mp_pairs) ;  //  ###   lock scans[],
                                        scan_pair( flow[i].ip[j], flow[i].ip[k],
TCP_PORT_SCAN, flow[i].pt[k] ) ; // ip[j] hit ip[k]:pt[k]
        pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                } }     }
                else { // not LOOSE
                        if((flow[i].scans >= SCAN_MAX) || (flow[i].flag[k][RST] ==
flow[i].pkts[k])){ // >SCANMAX or all RSTs
                                if(flow[i].flag[k][RST] >= (flow[i].pkts[k] /2 + 1))
{// if the other guy sent more resets
                                        host[h[j]].pt_scans++ ;
                                        list = 1 ; strcat( reason, "TCP_Pt_Scan ") ;
                                        flow[i].state |=  PROBE ;
                                        record_probe( h[j], flow[i].ip[k],
flow[i].pt[k]); // h[j] is source index of probe, 'k' is destination ip
        pthread_mutex_lock( &mp_pairs) ;  //  ### lock scans[],
                                        scan_pair( flow[i].ip[j], flow[i].ip[k],
TCP_PORT_SCAN, flow[i].pt[k] ) ; // ip[j] hit ip[k]:pt[k]
        pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                        } }
                }               // end - not LOOSE
        } // end - TCP check for Port Probes, not ATK_PROBE
} // end  j=0,1  k=1,0


for( j = 0 ; j < 2 ; j++ ){           // === 'TOUCHING' === mark hosts that send
data to high CI host
        k = 1 - j ;  // j,k =  0,1 and 1,0   ==========   'j' is this host, 'k' is
other host

        if( host[h[j]].alerts & ALARM_12W) { // ALARM-1 or -2 or WATCH_HOST    save
flow info
                list = 1 ;
                if( host[h[j]].alerts & ALARM_2) { // ALARM-2      ### HOSTS MARKED
ALARM_1 HERE ###

                        if( ( j == 1)  && host0 )  {  // j == 1
                                 // IP[0] marked ALARM-0 for sending data to bad guy
IP[1]

                                host[ host0 ].alerts |= ALARM_1 ;
                                host[ host0 ].alarm_t = t_here ;
                        }

                        if( j == 0 ) {     // j == 0
                                alarms0 = 1 ;    // set if ip[0] is bad guy
                        }
                }
        }

        if( alarms0 && (j == 1) && flow[i].bytes[j] ) { //j==1, ip[0] is bad, data
was sent
                host[h[j]].alerts |= ALARM_1 ; // IP[1] marked ALARM-0 for talking
to bad guy IP[0]
        }

        host0 = h[j] ;  // only used when j = 1, then -> ip[0]    // === END
'TOUCHING' ===



           // --- if ip[j] sent a packet that was not a RESET, then add to host[j]
client or server statistics
if(host[h[j]].last < flow[i].last)  host[h[j]].last = flow[i].last ;  // --  update
host 'last' time

if(( flow[i].pkts[j] > flow[i].flag[j][RST]) && (!(flow[i].state & ATK_PROBE))) { //
only if 'j' sent non-RES packets


        if( !udp ) {  //  TCP  ip[J] <-> host[h[j]]

                if(( flow[i].pkts[j] > 0 ) && ( flow[i].pkts[k] > 0) ) { // multiple
TCP packets
                        int common = 0 ;

                if( s[j] && (!c[j])) {   // "ip[j]" -> host[h[j]] is Server
                                host[h[j]].s_flows ++ ;

                                if( (flow[i].service < UDP_PORT_OFFSET) &&
(port_mask[ flow[i].service ]) )  common = 1 ;

                                if ((flow[i].service < LOW_PT_MAX) || common) {  //
server j is LO
                                        if(    !               host[h[j]].port_smin
) host[h[j]].port_smin = flow[i].service ;
                                        if( flow[i].service < host[h[j]].port_smin )
host[h[j]].port_smin = flow[i].service ;
                                        if( flow[i].service > host[h[j]].port_smax )
host[h[j]].port_smax = flow[i].service ;

                                        host[h[j]].server |= port_mask[
flow[i].service ] ;

if(f_verbose==4)        printf("  Server: %d, common: %d ", j, common ) ;
```

```
                                        if(( flow[i].pt_min < LOW_PT_MAX) &&
(flow[i].service != 22) && (flow[i].service != 37))  {
                                                if( ! ( (flow[i].pt_min ==
flow[i].service) && ((flow[i].service==53) || (flow[i].service == 109) ))) {
                                                        host[h[j]].bad_flow++ ;  //
DNS (53) and POP2 (109) talk as peer to peer on same ports.
                                                        list = 1 ;
                                                        strcat( reason, "LO-LO-S") ;
                                                        host[h[j]].alerts |=
LO_LO_CS ;  // do not alert on DNS or POP2  servers connecting
                                        host[h[j]].concern += LO_LO_CI ;
                                                }
                                        }
                                }  // end server port < UDP_PORT_OFFSET
                                else  {  // Server is HIGH
//                                                              host[h[j]].server |=
HIGH_PORT_SERVER ;  // appears to be high-port server
                                        host[h[j]].bad_flow++ ;

                                        if( flow[i].pt_min > LOW_PT_MAX ) {
                                                host[h[j]].alerts |= HI_HI_CS ; //
client k HI , port > 1023
                                    host[h[j]].concern += HI_HI_CI ;
                                                list = 1 ;strcat( reason, "HI-HI-S")
;
                                        }
                                        else {
                                                host[h[j]].alerts |= LO_HI_CS ;
                                    host[h[j]].concern += LO_HI_CI ;
                                                list = 1 ; strcat( reason,
"LO-HI-S") ;
                                        }
                                }

                                if( ! common)  {
                                        host[h[j]].server |= odd_bit ;//  bit for
odd service

                                        for( p =0 ; p < 10 ; p++ ) {
                                                if( host[h[j]].s_list[p] ==
flow[i].service ) break ;

                                                if( ! host[h[j]].s_list[p] ) {
                                                        host[h[j]].s_list[p] =
flow[i].service ;  // if uncommon port, add to list
                                                        break ;
                                                }
                                        }
                                } // end ! common
                        } // end ip[j] is server


                        if ( c[j] && (!s[j])) {// ip[j] -> host[h[j]] is client
                                int common = 0 ;
                                host[h[j]].c_flows ++ ;
                                if((flow[i].scans > SCAN_MAX) && (flow[i].pkts[j] >=
flow[i].pkts[k])) host[h[j]].pt_scans++ ;
                                if(    !                host[h[j]].port_cmin )
host[h[j]].port_cmin = flow[i].pt_min ;
                                if( flow[i].pt_min < host[h[j]].port_cmin )
host[h[j]].port_cmin = flow[i].pt_min ;
                                if( flow[i].pt_max > host[h[j]].port_cmax )
host[h[j]].port_cmax = flow[i].pt_max ;

                                if((flow[i].service < UDP_PORT_OFFSET) &&
(port_mask[ flow[i].service ])){
                                        common = 1 ;// if server is 'common' (in the
.server bit map) treat as 'LO' server
                                }
                                if(f_verbose==4)        printf("  Client: %d,
common: %d ", j, common ) ;

                                if ((flow[i].service < LOW_PT_MAX) || common) {  //
server k is LO
                                        host[h[j]].client |= port_mask[
flow[i].service ] ;  // assumes port -> service
                                        if( flow[i].pt_min >= LOW_PT_MAX )  list = 0
;  // NORMAL = DO NOT LIST IN FILE
                                        else {  // client j is low
                                                host[h[j]].alerts |= LO_LO_CS ; //
############################
                                    host[h[j]].concern += LO_LO_CI ;
                                                host[h[j]].bad_flow++ ;
                                                list = 1 ;
                                                strcat( reason, " LO_LO-C ") ;
                                        }
                        } else {  // server k is HI
//                                                                  host[h[j]].client |=
HIGH_PORT_CLIENT ;  // appears to be high-port server's client
                                        host[h[j]].bad_flow++ ;
                                        if( flow[i].pt_min >= LOW_PT_MAX ) {
                                                host[h[j]].alerts |= HI_HI_CS ; //
client k HI , port > 1023
                                    host[h[j]].concern += HI_HI_CI ;
                                                list = 1 ;strcat( reason, " HI-HI-C
") ;

                                        }
                                        else {
                                                host[h[j]].alerts |= LO_HI_CS ;
                                    host[h[j]].concern += LO_HI_CI ;
                                                list = 1 ;
                                                strcat( reason, " LO-HI-C ") ;

                                        }
                                }

                                if( ! common ) {
                                        host[h[j]].client |= odd_bit ;//  bit for
odd service

                                        for( p =0 ; p < 10 ; p++ ) {
                                                if( host[h[j]].c_list[p] ==
flow[i].service ) break ;

                                                if( ! host[h[j]].c_list[p] ) { // if
list[p] is empty, add pt[k] to it
                                                        host[h[j]].c_list[p] =
flow[i].service ;  // if uncommon port, add to list
                                                        break ;
                                                }
                                        }
                                } // end ! common
                        }  // ip[j], and host[h[j]], is client

                        if( c[j] && s[j] ) {
                                if( ! (flow[i].state & LOOSE) ) {
                                        host[h[j]].alerts |= PORT_SCAN2 ;
                                        host[h[j]].pt_scans++ ;
                                        host[h[j]].concern += PORT_SCAN2_CI ;
                                        flow[i].state |= PROBE ;
                                        list = 1 ;  strcat( reason, "C&S ") ;
                        }       }
                        if( ( ! c[j] ) && ( ! s[j]) ) {
                                if( ! (flow[i].state & LOOSE) ) {
                                        host[h[j]].u_flows++ ;
                                        host[h[j]].concern += UNKNOWM_CI ;
                                        host[h[j]].alerts |= NO_CS_SET    ;  // now
just a '.'
                                        list = 1 ;   strcat( reason, "NO-CS ") ;
                        }       }
                } //  end multiple pkts

        } // end TCP


        if( udp) {  // UDP
                if(f_verbose==4)        printf(" UDP Flow-%u ", j ) ;

                if( flow[i].pt[j] == 53 ) {
                         host[h[j]].server |= port_mask[ UDP_PORT_OFFSET + 53 ] ;
                         host[h[j]].dns_flows ++ ;
                }
                if( flow[i].pt[k] == 53 ) {
                        host[h[j]].client |= port_mask[ UDP_PORT_OFFSET + 53 ] ;
                        if( flow[i].pt[j] != 53 ) host[h[j]].dns_flows ++ ;
                }
                else if ( flow[i].pt[j] != 53 ) {  // neither port is 53

                        pps0 = flow[i].pkts[j] / ( 1 + flow[i].last - flow[i].start)
;
                        pps1 = flow[i].pkts[k] / ( 1 + flow[i].last - flow[i].start)
;

                        if( (pps0 + pps1 ) > 6 ) {
                                if(( flow[i].pkts[j] >> 2) > flow[i].pkts[k]) {
                                        host[h[j]].mm_s++; // host is one-wqy mm_s
                                        host[h[j]].server |=  mm_bit ;//  bit for
multimedia, psedo-port = UDP_PORT_OFFSET + 1
                                } else {  // not server
                                        if( (flow[i].pkts[k]  >> 2) >
flow[i].pkts[j]) {
                                                host[h[j]].mm_c++;// host is one-wqy
mm_c
                                                host[h[j]].client |= mm_bit ;//  bit
for multimedia
                                        } else  {  // not client either
                                                host[h[j]].mm_p++;// host is
multimedia peer (2-way)
                                                host[h[j]].client |= mm_bit ;//  bit
for multimedia
                                                host[h[j]].server |= mm_bit ;//  bit
for multimedia
                                        }  // end peer
                                }  // end not-server
                        }
//                                      else {  // check for GAME or TELEPH
//                                      }
                }  // not DNS, check for multimedia
        }       // end UDP
} // end if pkts[j] > 0
}  // end stats for host ip[j], j = 0, 1 & 'Touching'
flow[i].state |= CLASSIFIED ;  // flow has been classified
pthread_mutex_unlock( &mp_hosts) ;  //  ###  unlock hosts[]
} // end - not-classified and clearing flow

if(f_verbose==4)        printf("  Clear: %u, Reason: %s\n", clear, reason ) ;

if(clear) {             // ======================== END-OF-FLOW ANALYSIS (beyond
client-server) ============ //
                if( f_verbose == 3) {printf(" End-of-Flow Start. np:%u at
%u\n",np,*(&t_run));fflush( stdout);}

        if(!udp && (flow[i].state & ATTACK)){  // if TCP - check addition points for
half-open attack
                list = 1 ;
                for( j = 0 ; j <= 1 ; j++ ) {
                        k = 1 - j ;
                        if(( flow[i].pkts[j] > 16 ) && ( flow[i].flag[j][SYN] >
(0.75 * flow[i].pkts[j] )) &&
                                                        flow[i].flag[k][S_A] ) {  //
if half-open attack, 'j' attacking 'k'
                                if(( h[j] ) && ( host[ h[j] ].alerts == HO_ATTACK )
) {
                                        host[ h[j] ].concern += HO_ATK_PER_SYN *
(flow[i].flag[j][SYN] - 12 ) ;
                                }  // end - find_host() ok
                        }
                } // - end - for j = 0,1 and k = 1,0
        } // end - check for half-open (lots of SYN's evoking SYN-ACK's)

if(flow[i].state & ATTACK ) { list = 1 ; strcat( reason, " Attack") ;}
if(flow[i].state & PROBE  ) strcat( reason, " Probe" ) ; // don't set 'list' -too
many probe flows
} // end - end-of-flow analysis

if(clear && (list||f_all||f_demo||(flow[i].state & WATCH_FLOW))){  // == write to
flow-log =//


        unsigned int ip0, ip1 ;                     // option "a" (f_all == 1) logs
all flows
        unsigned char as0[16], as1[16] ;
        ip0 = flow[i].ip[0] ;
        ip1 = flow[i].ip[1] ;
if( f_verbose == 3) {printf(" List-Flow Start. i:%u at %u\n", i, *(&t_run)) ;
fflush( stdout);}

        n_fprint ++ ;           //  select flows to log
        n_flow_log ++ ; // cum
//      if( f_dotdec ) {                                //  cmd line flag 'd' -
print as dotted-decimal  --
sprintf( as0, "%3u.%3u.%3u.%3u",(ip0>>24) & 0xff, (ip0>>16) & 0xff, (ip0>>8) & 0xff,
ip0 & 0xff) ;
sprintf( as1, "%3u.%3u.%3u.%3u",(ip1>>24) & 0xff, (ip1>>16) & 0xff, (ip1>>8) & 0xff,
ip1 & 0xff) ;

        fprintf(log_flow, "%u\t %s\t %s\t %u\t %u\t %u\t %u\t %u\t",
                flow[i].state, as0, as1, flow[i].pt[0], flow[i].pt[1],
flow[i].service, flow[i].start, flow[i].last);
//} else {   // print as 8-hex
//      fprintf(log_flow, "%u\t %x\t %x\t %u\t %u\t %u\t %u\t %u\t",
//      flow[i].state, ip0, ip1, flow[i].pt[0], flow[i].pt[1], service,
flow[i].start, flow[i].last);
```

```
//}

fprintf(log_flow, "%u\t %u\t %x\t %u\t %u\t",
flow[i].bytes[0], flow[i].pkts[0], flow[i].flgs[0], flow[i].pt_min, flow[i].pt_max
);


if( flow[i].bin_norm[0] ) fprintf(log_flow, "%u\t",  (25 *
flow[i].binary[0])/flow[i].bin_norm[0]) ;
else fprintf(log_flow, "255\t") ;

fprintf(log_flow, "%u\t%u\t%u\t%u\t%u\t%u\t%u\t",
flow[i].flag[0][0], flow[i].flag[0][1], flow[i].flag[0][2], flow[i].flag[0][3] ,
flow[i].flag[0][4], flow[i].flag[0][5], flow[i].flag[0][6] );

fprintf(log_flow, "%u\t %u\t %x\t %u\t %u\t",
flow[i].bytes[1], flow[i].pkts[1], flow[i].flgs[1], flow[i].pt_min, flow[i].pt_max
);

if( flow[i].bin_norm[1] ) fprintf(log_flow, "%u\t",  (25 *
flow[i].binary[1])/flow[i].bin_norm[1]) ;
else fprintf(log_flow, "255\t") ;

fprintf(log_flow, "%u\t%u\t%u\t%u\t%u\t%u\t%u\t%s\n",
flow[i].flag[1][0], flow[i].flag[1][1], flow[i].flag[1][2], flow[i].flag[1][3],
flow[i].flag[1][4], flow[i].flag[1][5], flow[i].flag[1][6], reason );

}  // end of select to fprint(log_flow, ... )


        if( clear) {           //  ======= clear flow[i] ========== //
                if( flow[i].up)  flow[ flow[i].up ].down = flow[i].down ;  // if .up
not zero
                if( flow[ flow[i].down ].up   == i) flow[ flow[i].down ].up   =
flow[i].up ; // one is true
                if( flow[ flow[i].down ].root == i) flow[ flow[i].down ].root =
flow[i].up ;

                memset( (void *) & flow[i].ip[0] , '\0', (size_t) ( sizeof( struct
flow_db )  ) );
                flow[i].down = 0 ;
                if( flow[i].flag[1][7] != 0) printf("#### #### FLOW SLOT NOT BEING
ZEROED #### ####\n"); ;

                if(n_active > n_active_max) n_active_max = n_active ;
                n_active -- ;
        }  // end - clear flow

if( f_verbose == 3) {printf(" Flow_class 1 flow %u done. np:%u at %u\n", i, np,
t_run) ; fflush( stdout);}
                }// end of confirming (.down == 0)
pthread_mutex_unlock( &mp_flows) ;  //  ###  unblock process_pkts
        }  // end of (.down == 0) && remove flow
        }  // end i = 1 to SLOTS

        if(f_verbose) {
                fflush( log_flow ) ;
                printf("    t_run (s): %u, next web: %u,  Flows lost: %u, Flows
Ended: %u, Flows Listed: %u\n",
                                        *(&t_run), t_next_web - t_zero, n_full, flows_ended,
n_fprint) ;
        }
        if(( t_here >= t_next_web) && (running != 2)) {
                t_next_web += WEB_ALERT_INT ;
                web_alerts( ) ;                         // #####   #####    call
Web_Alerts
        }
        if(( t_here >= t_next_prof) && (running != 2)) {
                t_next_prof += WEB_ALERT_INT ;
                save_profiles( ) ;                      // #####   #####    call
save_profiles
        }
        this_hour =( (t_here / 3600) % 24 ) ;  // hour of the day, 1 - 23
        if( last_hour > 23 ) last_hour = this_hour ;

if(f_verbose >= 2) printf(" Restart_hour: %d Last_hour: %d  Day: %d, this_hour: %d,
minutes : %d\n",
restart_hour, last_hour, ((t_here / 86400) % 30), this_hour, (t_here / 60) % 60 ) ;

        if( (this_hour == restart_hour) && (last_hour != restart_hour) ) {
                i = save_profiles() ;
                printf("\n #########  Good Bye. Saved %u Profiles for Tomorrow.
###########\n\n", i ) ;
                suicide(  (t_here / 86400) % 30 ) ; // parameter is day 0-29, name
of log directory
        }
        last_hour = this_hour ; // for next check

        if( f_file)  {
                if( f_verbose >= 2 ) printf(" Classify_Flows about to sleep.  t_run
:%u\n", *(&t_run) ) ;
pthread_mutex_unlock( &mp_class_flow) ;  //  ###  unblock process_pkts
                sleep(1) ;
        }
        if( running == 2) break ; // run once to finish up

if( f_verbose == 3) {printf(" Flow-Class Bottom. np:%u at %u\n", np, t_run) ;
fflush( stdout);}
} // ============== =============== ===============end while(running)

if(f_verbose)  {
  printf("* Thread class_flow finishing at t: %u, np: %u\n",t_here - t_zero, np);
        fflush( log_flow ) ; fflush( stdout ) ;
}
        return ;
} // end class_flow()

//    ### SCAN_PAIR - KEEP STATISTICS BASED ON PROBER-TARGET IP ADDRESSES


unsigned long scan_pair(unsigned long ip0, unsigned long ip1, unsigned char type,
unsigned short port)
        {  //   type: 0 - empty, 1 - UDP, 2 - TCP, 8 - S-addr==D-addr
        unsigned long x, i, i_mark, n, index, scans_size = SCAN_SLOTS, min_type,
min_slot ;
        size_t t_here ;

        if( f_verbose == 3) {printf(" Scan_Pair Start. np:%u at %u\n", np,
*(&t_run)) ; fflush( stdout);}
        x =  ip0 ^ (ip1 << 6 ) ^ (ip1 >> 26)  ;/***  make index from DA, SA, port
***/
        index = x ;
        for (i = SCAN_SHIFT ; i < 32 ; i += SCAN_SHIFT ) {
                x = x >> SCAN_SHIFT ;
                index ^=  x ;           //  XOR bytes together
        }
        index  = index & SCAN_MASK ;  // limit to right SCAN_SHIFT bits
        index++ ;               // index can be 1 to 2^n +1

if( f_verbose > 1) printf("###scan_pair, np:%u, index: %u, type: %u, port: %u", np,
index, type, port ) ;
        t_here =  *(&t_run) + *(&z_sec) ;

                //  --- look for matching connection or first empty slot
        if( scans[index].root) {    //   root not empty, run through list, look for
matching scan
                i = scans[index].root ;
                do {
                        if((scans[i].ip0 != ip0) || (scans[i].ip1 != ip1)) {
                                if( (scans[i].up)) {
                                        i = scans[i].up;
                                } else {
                                        break ;   // break out of while(1) - end of
list, no match
                                }
                        } else { // slot for scan already exists
                                goto update_pair        ;
                        }
                } while( 1)   ; //  break out when end of list or match found

                i_mark = i ;      // index of last entry in list - NEW HOST ENTRY

                for( n = 0 ; n < SCAN_RANGE ; n++) {
                        i += 5 + n ;
                        if( i >= SCAN_SLOTS ) i = i - SCAN_SLOTS + 1;   // can not
use i = 0
                        if(scans[i].down == 0 ) {  //  empty, use for new entry
                                scans[i_mark].up = i ;
                                scans[i].down = i_mark ;
                                goto new_pair ;
                        }
                }
                return( 0 ) ;   // scan data sub-structure full
        }

        else {    //  scans[index].root == 0,  search for any empty slot for NEW
ENTRY

                i = index ;
                for( n = 0 ; n < SCAN_RANGE ; n++) {
                        i += 5 + n ;
                        if( i >= SCAN_SLOTS ) i = i - SCAN_SLOTS + 1;   // can not
use i = 0
                        if(scans[i].down == 0 ) {
                                scans[index].root = i ;
                                scans[i].down = index ;
                                goto new_pair ;  //
                        }
                }
                return( 0 ) ;   // scan data sub-structure full
        }
new_pair:
        n_scans++ ;
//      if( n > n_pr_search) n_pr_search = n ;  //  check hash efficiency
        scans[i].up    = 0 ;
        scans[i].ip0   = ip0 ;
        scans[i].ip1   = ip1 ;
        scans[i].start = scans[i].last   = *(&t_run) + t_zero ;
        scans[i].port[0] = port ;
        scans[i].type[0] = type ;
        scans[i].n_hits = 1 ;
        scans[i].n_ports = 1 ;
        scans[i].concern = 100 ;
        if(type > 8) scans[i].concern += 200 << (type - 8 ) ; // 9,10,11,12 ->
200,400,800,1600


if( f_verbose > 1) printf(",  new-pair, i:%u type: %d, port%d, concern: %u\n",
 i, (int) type, (int) port, scans[i].concern  ) ;

        return( i ) ;

update_pair:
min_type = 100 ;
        scans[i].last  = *(&t_run) + t_zero ;
        scans[i].n_hits ++ ;
        if( scans[i].n_ports < 16 ) {
                for(n = 0 ; n < 16 ; n++ ) { // fill a new slot or find a match
                        if( scans[i].type[n] < min_type ) {
                                min_type = scans[i].type[n] ;
                                min_slot = n ;
                        }
                        if( scans[i].type[n] == 0 ) {
                                scans[i].port[n] = port ;
                                scans[i].type[n] = type ;
                                scans[i].n_ports = n + 1 ;
                                break ;
                        }
                        if(scans[i].type[n]==type) {
                                if (scans[i].port[n] == port) { // already present
                                        if( scans[i].walk[n] < 0xff)
scans[i].walk[n] ++ ;
                                        break ;
                                }
                                if((port > 1023) && (port < (scans[i].port[n]+5))
&&(scans[i].walk[n]<255)){
                                        scans[i].port[n] = port ;  // port walking
                                        if( scans[i].walk[n] < 0xff)
scans[i].walk[n] ++ ;
                                        break ;
                        }       }
                }
        }    // breaks go here
        if( (n == 16) && (type > 6)) {
                if( type == scans[i].type[ min_slot ] ) {
                        scans[i].port[ min_slot ] = port ;
                }
                else {
                        if( type > scans[i].type[ min_slot ] ) {
                                scans[i].port[ min_slot ] = port  ;
                                scans[i].type[ min_slot ] = type  ;
        }       }       }


        scans[i].concern += 100 ;
        if(type > 8) scans[i].concern += 200 << (type - 8 ) ; // 9,10,11,12 ->
200,400,800,1600

        if( f_verbose > 1) printf(",  update-pair, i:%u, pair hits: %u, slots
filled: %u, concern: %u \n",
       i, scans[i].n_hits, scans[i].n_ports, scans[i].concern) ;
                if( f_verbose == 3) {printf(" Scan_Pair End. np:%u at %u\n", np,
*(&t_run)) ; fflush( stdout);}

return( i ) ;
} // end of scan_pair



unsigned long find_host( unsigned long ip, unsigned int make)
        {// returns host[i] unsigned Index, HOSTS MUST BE LOCKED
        unsigned long x, i, i_mark, n, index, scans_size = HOST_SLOTS, unlock = 0 ;

        x =  ip ^ (ip <<  6) ;/***  make index from ip ***/
        index = x ;
        for (i = HOST_SHIFT ; i < 32 ; i += HOST_SHIFT ) {
                x = x >> HOST_SHIFT ;
                index ^=  x ;           //  XOR bytes together
        }
        index  = index & HOST_MASK ;  // limit to right HOST_SHIFT bits
        index++ ;               // index can be 1 to 2^n +1

        if( ! pthread_mutex_trylock( &mp_hosts ) ) {  // this will lock mutex if not
locked already
printf(" ### find_host() without mutex m_host being set. t: %u, Running:
%u\n",*(&t_run),running);
                unlock = 1 ;
        }
#ifdef CHECK_INDEX
        if ( index >=  HOST_SLOTS ) {
                printf("index bigger than HOST_SLOTS (%u > %u) at record %u\n",
index, HOST_SLOTS, *(&np));
                exit(-1) ;
        }
#endif
                //  --- look for matching HOST or first empty slot

        if( host[index].root) {    //   root not empty, run through list, look for
matching scan
                i = host[index].root ;
                do {
                        if(host[i].ip != ip) {
                                if( (host[i].up)) {
                                        i = host[i].up;  //  loop again
                                } else {
                                        break ;   // break out of while(1) - end of
list, no match
                                }
                        } else {
                                host[i].last = *(&t_run) + t_zero ;  // this is
"update_host:"
if( unlock ) pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
                                return( i ) ;   // slot for scan already exists

                        }
                } while( 1)   ; //  break out when end of list, match found ->
return

                i_mark = i ;      // index of last entry in list - NEW HOST ENTRY

                if( make == 0) return( 0 ) ; // !make -> do not make a new host
entry

                for( n = 0 ; n < HOST_RANGE ; n++) {
                        i += 5 + n ;
                        if( i >= HOST_SLOTS ) i = i - HOST_SLOTS + 1;   // can not
use i = 0
                        if(host[i].down == 0 ) {  //  empty, use for new entry
                                host[i_mark].up = i ;
                                host[i].down = i_mark ;

                                goto new_host ;
                        }
                }
                printf("H") ;
if( unlock ) pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
                return( 0 ) ;   // Host data sub-structure full
        }

        else {    //  host[index].root == 0,  search for any empty slot for NEW
ENTRY
                if( make == 0 ) return( 0 ) ; // !make -> do not make a new host
entry
                i = index ;
                for( n = 0 ; n < HOST_RANGE ; n++) {
                        i += 5 + n ;
                        if( i >= HOST_SLOTS ) i = i - HOST_SLOTS + 1;   // can not
use i = 0
                        if(host[i].down == 0 ) {
                                host[index].root = i ;
                                host[i].down = index ;
                                goto new_host ;
                        }
                }
                printf("H") ;
if( unlock ) pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
                return( 0 ) ;   // Host data sub-structure full
        }

new_host:
        host[i].up   = 0 ;
        host[i].ip = ip ;
        host[i].start = host[i].last = *(&t_run) + t_zero ;
        for ( n = 0 ; n < ln_max ; n++) {
                if( ! ( local_mask[ n ] & ( local_net[ n ] ^ ip ))) {
                        host[i].alerts |= LOCAL_HOST ;  // set local-host bit
                        active_locals++ ;
                        break ;
                }
        }
        n_host++ ;
        if( n > n_pr_search) n_pr_search = n ;  //  check hash efficiency
        return( i ) ;
if( unlock ) pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
} // end of find_host()


void    print_host(FILE *xfile, unsigned long h, unsigned long v)  //  NOT PRESENTLY
USED
{
        unsigned long j, ip, n, x, a0, a1, a2, a3 ;
        char as0[16] ;
        ip = host[h].ip ;
sprintf( as0, "%3u.%3u.%3u.%3u",(ip>>24) & 0xff, (ip>>16) & 0xff, (ip>>8) & 0xff, ip
& 0xff) ;
        for( j = 0 ; j < strlen(as0) ; j++ ) if( as0[j] == ' ') as0[j] = '0' ;

        a1 = 64 * host[h].pt_scans ;   //  MUST MATCH CALCULATION IN 'web_alerts()'
        if( host[h].alerts & PORT_SCAN2    ) a1 += 400 ;
        if( host[h].alerts & PT_SCAN_ALERT ) a1 += 400 ;
        if( host[h].alerts & HI_HI_CS      ) a1 += 200 ;
        if( host[h].alerts & LO_HI_CS      ) a1 += 800 ;
        if( host[h].alerts & LO_LO_CS      ) a1 += 200 ;
        a2 =  8 * host[h].no_con_t  + 8 * host[h].rejects  + 1 * host[h].pings + 4 *
host[h].traces
                                +  8 * host[h].bad_pkts  + 8 * host[h].bad_flow + 8
* host[h].u_flows;
        a3 = 0 ;                        //  application related
//      if( host[h].server & IRC ) a3 += 200 ;
//      if( host[h].client & IRC ) a3 += 100 ;
        a0 = a1 + a2 + a3 ;


fprintf(xfile, " %u\t%u\t%u\t%u\t%s\t %u\t %u\t", a0, a1, a2, a3, as0,
host[h].start, host[h].last) ;
fprintf(xfile, " %u\t %u\t", host[h].bytes_in, host[h].bytes_ot) ;
fprintf(xfile, " %u\t %u\t %u\t %u\t", host[h].pkts_in, host[h].pkts_ot,
host[h].port_smin, host[h].port_smax);
fprintf(xfile, " %u\t %u\t %x\t", host[h].port_cmin, host[h].port_cmax,
host[h].server );
fprintf(xfile, " %x\t %x\t %u\t %u\t", host[h].client, host[h].alerts,
host[h].pt_scans, host[h].resets );
fprintf(xfile, " %u\t %u\t %u\t", host[h].rejects, host[h].no_con_t,
host[h].dns_flows);
fprintf(xfile, " %u\t %u\t %u\t %u\t", host[h].udp_bytes, host[h].mm_s,
host[h].mm_c, host[h].mm_p);
fprintf(xfile, " %u\t %u\t %u\t", host[h].s_flows, host[h].c_flows, host[h].u_flows
);
fprintf(xfile, " %u\t %u\t %u\t %u\t", host[h].pt_scans, host[h].bad_pkts,
host[h].pings, host[h].traces);

fprintf(xfile, "S: ");
        x = host[h].server ; n = 0 ;
        while(( x > 0) && (n < pn_max)) {
           if( x & 1) fprintf(xfile,"%s ", port_name[ n ] ) ;
           n++ ; x = x >> 1 ;
        }
fprintf(xfile, "- C: ");
        x = host[h].client ; n = 0 ;
        while(( x > 0) && (n < pn_max))  {
           if( x & 1) fprintf(xfile,"%s ", port_name[ n ] ) ;
           n++ ; x = x >> 1 ;
        }
        fprintf(xfile, "\n") ;
}  //  end of print_host()



unsigned long dots_int( char * p )  { // dotted-decimal to integer

        unsigned int v, j, ipv ;
        char *sp1, *sp2, buf[120]  ;

        strcpy(buf, p) ;
        buf[15] = 0 ;
        v = 0 ;
        j = 0 ;
        sp1 = sp2 = buf ;  // beginning of string "149.77.8.2"
        while (j< 5) {
            j++ ;
                sp2 = strstr(sp1,".") ;
                if(sp2) {                     // there's a dot
                        sp2[0] = 0 ;                // sp1 -> "177"
                        v = v * 256 + atoi( sp1 ) ;
                        sp1 = &sp2[1] ;             // sp1 -> "77.8.2"
                } else {
                        ipv = v*256 + atoi( sp1) ; // sp1 -> "2"
                        break ; //  out of while loop
                }
                if( (atoi(sp1) > 255) || (atoi(sp1) < 0 ) ) j = 10 ; // -> error,
below
        }
        if( j != 4 ) {
                printf(" Can not parse IP: '%s', Quitting\n", p ) ;
                usage() ;
        }
        return( ipv ) ;
}  // end of dots_int()



void web_alerts( void ) { // ====  run every 1200 s, only lock  hosts
        char *sp, s1[20], s2[400], s3[400], s4[100], web_ts[64], wts[128], ts[16] ;
        FILE *web_alt0, *web_alt1, *web_traf0, *web_traf1, *web_hosts, *web_info,
*lc_th_file, *web_alarms;
        FILE *web_scans ;
        unsigned int h, i, j, n, n0=0, n1=0, n2=0, n3=0, nh=0, nf=0, n_scan_prt,
n_host_drop, drop_hosts ;
        unsigned int x, a, a0, a1, a2, a3, pu, file_fail, error_count, odd_bit, traf
;
        time_t       t_1, t_2, t_here, t_tab, t_cut ;
        struct tm               *w_time ;
        int m, local, dst ;

        if( f_verbose) printf(" 'Web_alert' started at t_run: %u s, pkts: %u\n",
*(&t_run), np) ;
        if( f_verbose > 1 ) fflush( stdout ) ;
        if( *(&t_run) > LOOSE_PERIOD ) f_loose = f_splitpath ; // continue loose
classify if split_path
         odd_bit = port_mask[ 2 * UDP_PORT_OFFSET + 4] ; //  bit for odd service,
psedo-port = 2052

        if( f_file) t_here = *(&t_run) + t_zero ;
  else  t_here = time( NULL ) ; // real time

pthread_mutex_lock( &mp_bs) ;  //  ### lock bs[]
        if(  t_here  > (t_bs_last + TRAF_TABLE_MIN ) ) {
                bt ++ ; // bs[ bt + 1 ] may not be reliable in another thread, also
bt may be > TRAFFIC_VALUES
                if( bt == TRAFFIC_VALUES ) bt = 0 ;
//              bs[ bt ].bytes_in = bs[ bt ].bytes_out = bs[ bt ].bytes_loc = bs[ bt
].bytes_oo = 0 ;
                memset( (void *) &bs[ bt ] , '\0', (size_t) ( sizeof( struct
byte_table ) ) );
                t_bs_last = t_here - 1 ;
        }

                bs[ bt ].t          = t_here ;
                bs[ bt ].bytes_in  += bytes_in_cnt  ;
                bs[ bt ].bytes_out += bytes_out_cnt ;
                bs[ bt ].bytes_loc += bytes_loc_cnt ;
                bs[ bt ].bytes_oo  += bytes_oo_cnt  ; //----------> Record Traffic
Load for Web Table
                bs[ bt ].bytes_bc  += bytes_bc_cnt  ;
                bs[ bt ].bytes_mcn += bytes_mcn_cnt ;
                bs[ bt ].bytes_mco += bytes_mco_cnt ;
                bs[ bt ].bytes_bad += bytes_bad_cnt ;


                if(bs[ bt ].bytes_in  > bps_in_max )  bps_in_max  = bs[ bt
].bytes_in  ;
                if(bs[ bt ].bytes_out > bps_out_max)  bps_out_max = bs[ bt
].bytes_out ;
                if(bs[ bt ].bytes_loc > bps_oo_max )  bps_oo_max  = bs[ bt
].bytes_loc ;
                if(bs[ bt ].bytes_oo  > bps_loc_max)  bps_loc_max = bs[ bt
].bytes_oo  ;
                if(bs[ bt ].bytes_mcn > bps_mcn_max)  bps_mcn_max = bs[ bt
].bytes_mcn ;
                if(bs[ bt ].bytes_mco > bps_mco_max)  bps_mco_max = bs[ bt
].bytes_mco ;
                if(bs[ bt ].bytes_bc  > bps_bc_max )  bps_bc_max  = bs[ bt
].bytes_bc  ;
                if(bs[ bt ].bytes_bad > bps_bad_max)  bps_bad_max = bs[ bt
].bytes_bad ;

                bytes_in_cnt  = bytes_out_cnt = bytes_loc_cnt = bytes_oo_cnt  = 0 ;
                bytes_mcn_cnt = bytes_mco_cnt = bytes_bc_cnt  = bytes_bad_cnt = 0 ;

pthread_mutex_unlock( &mp_bs) ;  //  ### unlock bs[]

//if(f_file) {
//      t_next_web = WEB_ALERT_INT ;     // left over from when this was a separate
thread
//}
//else {
//      unsigned int wai = WEB_ALERT_INT ;
//      t_next_web = t_zero + 0.5 * wai ;
//      t_next_web = t_next_web + wai - (t_next_web % wai) + 1 ; // moves to nearest
even value +1
//}  // so new value traffic graph will complete before this runs

//while( running)
//      sleep( 10) ; // in case a file problem causes a "continue"

//      while(( t_here < t_next_web) && (running == 1)) {
//        if( f_file) t_here = *(&t_run) ;
//        else t_here = time( NULL ) ;
//              sleep(10) ;  //  check time and 'running' every 10 seconds
//      }

        if( running == 0 ) return ; // break ;
        t_1 = t_here ;
        t_2 = time( NULL ) ;
        w_time = localtime( & t_1) ;
        dst = w_time->tm_isdst ; // >0 if daylight savings time
        alarm_lines = 0 ;
pthread_mutex_unlock( &mp_bs) ;  //  ### lock hosts[]

pthread_mutex_lock( &mp_hosts) ;  //  ### unlock bs[]
        read_thresholds( ) ; // update thresholds from file lc_thresholds.txt
pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]

  strcpy( web_ts, "Web Prepared: " ) ;
        strcat( web_ts, asctime( w_time )) ;
        web_ts[ 38 ] = 0 ; // delete \n from asctime()
        strcpy( wts, monitor_start ) ;
        strcat( wts, web_ts ) ;
if(f_verbose) { printf("* Web_alert operating at t: %u, np: %u, %s\n", t_here -
t_zero, np, web_ts) ;
fflush( stdout) ;       }

//  ================   START OF PROBE TABLE ===================//

        if( (web_scans = fopen("web_scans.txt","w") )==NULL) {  //  -------OPEN
OUTPUT FILE web_scans.txt
                printf("\nLog-Scans File 'web_scans.txt' Can Not Be Opened\n\n") ;
        }
        else {  //  file 'web_scans' opened OK

        register int i, j, n ;
        n_scan_prt = 0 ;
        for( i = 0 ; i < 15 ; i++ ) scin[ i ] = 0 ;

if( f_verbose == 2 ) {printf(" --- Scan-Pairs about to begin.\n") ; fflush( stdout )
; }

        for( i = 1 ; i < SCAN_SLOTS ; i++ ) { //    ### WRITE SCAN-PAIRS  ###
                register unsigned int ip0, ip1;
                char as0[20], as1[20], as[30], as4[30], pr_str[500], x_str[50] ;
                if( scans[i].down)  { //  select pairs to fprint
                        register int j, k, tf, done ;
                        int delete_pair = 0, display_pair = 0 ;

pthread_mutex_lock( &mp_pairs) ;  //  ### lock scans[],
                        if( ( scans[i].last + 1200 ) > t_here ) tf = scans[i].last +
1200 - t_here ; // time factor for display
                        else tf = 0 ;

                        if(((scans[i].concern + tf ) < scans_cut )) {
                                delete_pair = 1 ;
                        }
                        else {
                                for( k = 1 ; k < 15 ; k++ ) {  //  put scans with
values of concern into bins
                                        if( ( scans[i].concern + tf ) < cix[k]) {
                                                scin[k]++ ;
                                                break ;
                                        }
                                }
                        } // end - not cut


                        if(((scans[i].concern + tf ) > scans_pr_cut )) display_pair
= 1 ;

        if( f_verbose == 2 ) printf(" scans_cut: %u, scans_pr_cut %u,
scans[%u].concern: %u\n",
         scans_cut, scans_pr_cut, i, scans[i].concern ) ;

         if( delete_pair || display_pair ) {
                                ip0 = scans[i].ip0 ;
                                ip1 = scans[i].ip1 ;
        sprintf( as0, "%3u.%3u.%3u.%3u",(ip0>>24) & 0xff, (ip0>>16) & 0xff, (ip0>>8)
& 0xff, ip0 & 0xff) ;
        sprintf( as1, "%3u.%3u.%3u.%3u",(ip1>>24) & 0xff, (ip1>>16) & 0xff, (ip1>>8)
& 0xff, ip1 & 0xff) ;
                                for( j = 0 ; j < strlen(as0) ; j++ ) if( as0[j] == '
') as0[j] = '0' ; //  target ip0
                                for( j = 0 ; j < strlen(as1) ; j++ ) if( as1[j] == '
') as1[j] = '0' ;  // prober ip1

                sprintf(pr_str, "%u\t%u\t%s\t%s\t%u\t%u\t", scans[i].last,
scans[i].start, as0, as1,
                   scans[i].n_hits, scans[i].n_ports );
                        for( j = 0 ; j < 16 ; j++ ) {
                                x_str[0]  = 0 ; done = 0 ;
                                switch ( scans[i].type[j] ) {
        case  1:  sprintf(x_str, "UDP_R-%u", scans[i].port[j])       ;break ; //
UDP_PROBE
        case  2:  sprintf(x_str, "TCP_R-%u", scans[i].port[j])       ;break ; //
TCP_PROBE
        case  3:  sprintf(x_str, "Short_UDP-%u" , scans[i].port[j]) ;break ; //
SHORT_UDP_SCAN
        case  4:  sprintf(x_str, "Src=Des-%u", scans[i].port[j])    ;break ; //
BOOMERANG
        case  5:  sprintf(x_str, "Ping")                            ;break ; //
PING_SCAN
        case  6:  sprintf(x_str, "ICMP_TO" )                        ;break ; //
ICMP_TO
        case  7:  sprintf(x_str, "TCP_TO-%u", scans[i].port[j])     ;break ; //
TCP_TO
        case  8:  sprintf(x_str, "UDP_TO-%u", scans[i].port[j])     ;break ; //
UDP_TO
        case  9:  sprintf(x_str, "Bad_TCP-%u", scans[i].port[j])    ;break ; //
BAD_PKT_TRACE
        case 10:  sprintf(x_str, "Multi_Pt-%u", scans[i].port[j])   ;break ; //
TCP_PORT_SCAN
        case 11:  sprintf(x_str, "Addr_Scan-%u", scans[i].port[j])  ;break ; //
TCP_ADDR_SCAN
        case 12:  sprintf(x_str, "HO_ATTACKn-%u", scans[i].port[j]) ;break ; //
HALF_OPEN
        default : done = 1 ;
                                        }
                                if( done ) break ;
                                strcat( pr_str, x_str ) ;
                                if( scans[i].walk[j] > 0 ) {
                                        sprintf(x_str, "(%u) ",  scans[i].walk[j] )
;

                                        strcat( pr_str, x_str) ;
                                }
                                else  strcat(pr_str, " " ) ;
                        } // end j = 0, 1,2, ...
                        strcat(pr_str, "\n") ;
                }
                if( delete_pair ) { //  if( delete_pair )        //  ======= clear
scan[i] ==========//
                        fprintf(log_pair,"%u\t%s",scans[i].concern, pr_str) ;  //
write to log before deleting

        if( scans[i].up )  scans[ scans[i].up ].down = scans[i].down ;  // if .up
not zero
        if( scans[ scans[i].down ].up   == i) scans[ scans[i].down ].up   =
scans[i].up ; // one is true
        if( scans[ scans[i].down ].root == i) scans[ scans[i].down ].root =
scans[i].up ;

                                memset( (void *) & scans[i].ip0 , '\0', (size_t) (
sizeof( struct ip_pair )  ) );
                                n_scans -- ;
                        }
pthread_mutex_unlock( &mp_pairs) ;  //  ### unlock scans[],
                        if(display_pair) {
                                fprintf(web_scans,"%10u\t%s", scans[i].concern + tf,
pr_str) ; // sort on concern + tf
                                n_scan_prt ++ ;
                        }
                }
        } // end for i
        fclose ( web_scans ) ;
}  // end - if fopen web_alerts.txt

if( f_verbose == 2 ) {
        printf(" --- Scan-Pairs done. Display: %d, Show > %d, Delete< %d\n",
                        n_scan_prt, scans_pr_cut, scans_cut);
        fflush( stdout )        ;
}

//  ================   END OF PROBE TABLE ===================//


        t_diff = t_here - t_wa_last ; // for web_alert bps cal
        t_wa_last = t_here ;
        if(t_diff < 100) t_diff = 100 ;  // may be small at first and when running
== 2

        web_traf_2byte = (web_traf_2 * t_diff) >> 3 ; // *dt/8

        if((web_info = fopen ("web_info.txt","wb"))==NULL) {   //  OPEN web_info
FILE
                printf("File 'web_info.txt' Can Not Be Opened to Write, t: %u\n", *
(&t_run)) ;
                 file_fail = 1 ; goto close_files ;
        }

        if((web_alt0 = fopen ("web_alerts0.txt","wb"))==NULL) {   //  OPEN
web_alerts0 FILE
                printf("File 'web_alerts0.txt' Can Not Be Opened to Write, t: %u\n",
* (&t_run)) ;
                file_fail = 2 ; goto close_files ;
        }

        if((web_alt1 = fopen ("web_alerts1.txt","wb"))==NULL) {   //  OPEN
web_alerts1 FILE
                printf("File 'web_alerts1.txt' Can Not Be Opened to Write, t: %u\n",
* (&t_run)) ;
                file_fail = 3 ; goto close_files ;
        }

        if((web_traf0 = fopen ("web_traf0.txt","wb"))==NULL) {   //  OPEN web_traf0
FILE
                printf("File 'web_traf0.txt' Can Not Be Opened to Write, t: %u\n",*
(&t_run)) ;
                file_fail = 4 ; goto close_files ;
        }

        if((web_traf1 = fopen ("web_traf1.txt","wb"))==NULL) {   //  OPEN web_traf1
FILE
                printf("File 'web_traf1.txt' Can Not Be Opened to Write, t: %u\n",*
(&t_run)) ;
                file_fail = 5 ;  goto close_files ;
        }

        if((web_hosts = fopen ("web_hosts.txt","wb"))==NULL) {   //  OPEN web_hosts
FILE
```

```
                printf("File 'web_hosts.txt' Can Not Be Opened to Write, t: %u\n",*
(&t_run)) ;
                file_fail = 6 ; goto close_files ;
        }

        if((web_alarms = fopen ("web_alarms.txt","wb"))==NULL) {    //  OPEN
web_alarm FILE
                printf("File 'web_alarms.txt' Can Not Be Opened to Write, t: %u\n",*
(&t_run)) ;
                file_fail = 7 ; goto close_files ;
        }

//      t_next_web += WEB_ALERT_INT ; // no 'continues' past here

        fprintf(web_info,"%u\t%u\t%u\t%d\t%u\t%u\t%u\t%u\n", pn_max, active_locals,
                                n_host - active_locals, n_scan_prt, *(&web_traf_2),
*(&web_alert_2), web_traf_2byte, dst);
        fprintf(web_info,"%u\t%u\t%u\t%u\t%s\t%u\t%u\t%u\t%u\n", bps_in_max,
bps_out_max,
                                bps_loc_max, bps_oo_max, wts, bps_mcn_max,
bps_mco_max, bps_bc_max, bps_bad_max );

fprintf(web_info,"Time\tOut-In\tIn-Out\tIn-In\tOut-Out\tOut-MC\tIn-MC\tB'cast\tBad\n
" ) ;

pthread_mutex_lock( &mp_bs ) ;  //  ### lock bs[]
        m = bt ;
        if( m >= TRAFFIC_VALUES) m -= TRAFFIC_VALUES ;  // prevents thread sync
problem
        m++ ;
        for( j = 0 ; j < TRAFFIC_VALUES ; j++ ) {  // j is counter
                m-- ;
                if(m < 0 ) m += TRAFFIC_VALUES ;  // m is index
                if( bs[m].t == 0) break ;
                w_time = localtime( & bs[m].t ) ;  // w_time points to 'tm'
structure
        strftime(ts, 16, "%H:%M", w_time) ; // t_table is formatted into string ts
fprintf(web_info,"%s\t%u\t%u\t%u\t%u\t%u\t%u\t%u\t%u\n",ts, bs[m].bytes_in,
bs[m].bytes_out,
    bs[m].bytes_loc, bs[m].bytes_oo, bs[m].bytes_mcn, bs[m].bytes_mco,
bs[m].bytes_bc, bs[m].bytes_bad);
        }
pthread_mutex_unlock( &mp_bs) ;  //  ### unlock bs[]

        fprintf(web_hosts,"0000\tActive Local Hosts:
%u\t%s\t\t\t\t\tServers\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\t\tClients\n"
                                 , active_locals, wts);
        fprintf(web_hosts,"0001\tHost IP\tConcern\tAlerts\tBytes-In\tBytes-Out\t\%
UDP");
        for( j = 0 ; j < pn_max ; j++ ) fprintf(web_hosts,"\t%s", port_name[ j ] );
// servers
        for( j = 0 ; j < pn_max ; j++ ) fprintf(web_hosts,"\t%s", port_name[ j ] );
// clients
        fprintf(web_hosts,"\n");

        fprintf(web_alt0,"4294967295\tLocal Hosts with High Concern
Indices\t%s\t%u\n",wts,*(&web_alert_2));
        fprintf(web_alt0,"4294967294\tConcern\tHost IP\tBytes-In\tBytes-Out\t\%
UDP\tServers\tClients\tAlerts\n");

        fprintf(web_alt1,"4294967295\tOutside Hosts with High Concern
Indices\t%s\t%u\n",wts,*(&web_alert_2));
        fprintf(web_alt1,"4294967294\tConcern\tHost IP\tBytes-In\tBytes-Out\t\%
UDP\tServers\tClients\tAlerts\n");

        fprintf(web_traf0,"4294967295\tLocal Hosts with Recent High
Traffic\t%s\t%u\t%u\t%u\n",
              wts, *(&web_traf_2), t_diff, running);
        fprintf(web_traf0,"4294967294\tTotal\tBit/s-In\tBit/s-Out\t\% UDP\tHost
IP\tConcern\tServers\tClients\tAlerts\n");

        fprintf(web_traf1,"4294967295\tOutside Hosts with Recent High
Traffic\t%s\t%u\t%u\t%u\n",
              wts, *(&web_traf_2) ,t_diff, running);
        fprintf(web_traf1,"4294967294\tTotal\tBit/s-In\tBit/s-Out\t\% UDP\tHost
IP\tConcern\tServers\tClients\tAlerts\n");

        memset( (void *) cin , '\0', (size_t) ( 30 * sizeof( long )  ) ); // clear
cin[ ]
        memset( (void *) tin , '\0', (size_t) ( 30 * sizeof( long )  ) );

        n_host_drop = 0 ;
        drop_hosts = ( n_host > HOST_LIMIT) ;  // TRUE if too many hosts in database
        if( f_file ) t_cut = t_here - 200 ; else t_cut = t_here - 1800 ;  // do not
drop newbies

//      memset( (void *) temp, '\0' , (size_t) (128 * sizeof( long)) ) ;  // DeBuG

        for( h = 1 ; h < HOST_SLOTS ; h++ ) {
                if(host[h].down) { //  select hosts to fprint

pthread_mutex_lock( &mp_hosts) ;  //  ### lock hosts[]

        if( host[h].down ) { //  select hosts to fprint - confirm after lock

                register int list_ci, list_traf  ;

if( f_verbose == 2 ) {printf(" --- New Host. h:%u, ip: %8x\n", h, host[h].ip) ;
fflush( stdout ) ; }
                list_ci = list_traf = 0 ;

                a1 = host[h].concern ;     //  ==============  Calculate Concern
Index (CI = a0)  ======== //

                a2 =  8 * host[h].no_con_t  + 8 * host[h].rejects ; // treat pings
and traces as probes
                                            +  8 * host[h].bad_pkts  + 8 *
host[h].bad_flow + 8 * host[h].u_flows;
                a3 = 0 ;                     //  application related
//              if( host[h].server & IRC ) a3 += 200 ;
//              if( host[h].client & IRC ) a3 += 100 ;
                a0 = a1 + a2 + a3 ; // this is the CI
                                                                           //  ###
### ALARMS 2 and 3 SET HERE ### ###
                traf = host[h].bytes_in_pp + host[h].bytes_ot_pp ;
                local = host[h].alerts & LOCAL_HOST ;

                if( a0 > web_alert_2)  {  // if CI greater than limit from
lc_thresholds.txt
                        list_ci = 1 ;  // list even if alarm shut off
                        if( ! ( host[h].alerts & NO_ALARM )) {  // if alarms not
shut off
                                host[h].alerts |= ALARM_2 ;  // set bit, cause email
to NetAdmin
                                host[h].alarm_t = t_here ;   // mark time to avoid
too-frequent alerts
```

```
                        }
                        else host[h].alerts &= (~ALARM_2) ;  //  alarm shut off, if
on
                }
                else host[h].alerts &= (~ALARM_2) ;  // clear ALARM_2 if web_alert_2
increased

                if( traf >= web_traf_2byte ) {
                        host[h].alerts |= TRAF_ALARM ;  // send email to Net Admin
                        host[h].alarm_t = t_here;
                        list_traf = 1 ;
                }
                else host[h].alerts &= (~TRAF_ALARM) ;  // clear TRAF_ALARM if
web_traf_2byte increased

                host[h].port_scan[0] = host[h].port_scan[1] = 0 ; // clear
short-term port scan
                host[h].scan_cntr[2]  = 0 ;


if( f_verbose == 2 ) {printf(" --- Alarms Set or Cleared.  traf: %u\n", traf) ;
fflush( stdout ) ; }

                if( local ) {                                   // local host lists
                                if( (a0 >= web_alert0_0)  || (host[h].alerts &
ALARM_12) || f_demo )   list_ci   = 1 ;
                                if( (traf >= web_traf0_0) || f_demo )
                list_traf = 1 ;
                }
                else {
// outside host lists
                                if( (a0 >= web_alert1_0) || (host[h].alerts &
ALARM_12) || f_demo )   list_ci   = 1 ;
                                if( (traf > web_traf1_0) || f_demo )
               list_traf = 1 ;
                }


                for( i = 1 ; i < 15 ; i++ ) {  //  put hosts with values of a0 into
bins to set web_alert_0
                        if( a0 < cix[i]) {
                                cin[!local][i]++ ;
                                break ;
                        }
                }

                for( i = 1 ; i < 15 ; i++ ) { //  put hosts with values of traf into
bins to set web_traf_0
                        if( traf < tix[i]) {
                                tin[!local][i]++ ;
                                break ;
                        }
                }
if( f_verbose == 2 ) {printf(" --- Going to Calculate UDP\n") ; fflush( stdout ) ; }

                if( host[h].bytes_in_mx < host[h].bytes_in_pp) host[h].bytes_in_mx =
host[h].bytes_in_pp ;
                if( host[h].bytes_ot_mx < host[h].bytes_ot_pp) host[h].bytes_ot_mx =
host[h].bytes_ot_pp ;

                if(  host[h].bytes_in   + host[h].bytes_ot ) {
                        if( host[h].udp_bytes > 42000000 ) {  // 100 * will cause
overflow
```

```
                                pu = (100 *(host[h].udp_bytes >>
8))/((host[h].bytes_in  + host[h].bytes_ot)>>8+1); // %UDP
                        } else {
                                pu = (100 * host[h].udp_bytes)/( host[h].bytes_in
+ host[h].bytes_ot + 1); // %UDP
                        }
                } else {
                        if( host[h].udp_bytes) pu = 100 ; else pu = 0 ;
                }

                a = host[h].ip ;
          sprintf( s1, "%3u.%3u.%3u.%3u", a>>24, (a>>16) & 0xff, (a>>8) & 0xff, a &
0xff ) ; // IP ddec
          for( j = 0 ; j < strlen(s1) ; j++ ) { if( s1[j] == ' ') s1[j] = '0' ; }


if((local)|| (host[h].alerts & ALARM_12) || list_ci || list_traf || f_demo)
                        {
if( f_verbose == 2 ) {printf(" --- Goint to List\n") ; fflush( stdout ) ; }

                                if(host[h].pings    > 100 ) host[h].alerts |=
PING_ALERT ;
                                if(host[h].traces   > 100 ) host[h].alerts |=
TRACE_ALERT ;
                                if(host[h].rejects  > 50 ) host[h].alerts |=
REJECT_ALERT ;
                                if(host[h].bad_pkts > 10 ) host[h].alerts |=
PKT_ALERT   ;
                                if(host[h].pt_scans >  5 ) host[h].alerts |=
PT_SCAN_ALERT ;

                                x = (host[h].alerts >> 1) ; n = 1 ; sp = s4 ; s4[0]
= 0 ; // s4 - alert list, skip '0' local
                                while(( x > 0) && (n < max_alert)) {
                                if( x & 1) {
                                        sprintf(sp,"%s ", alert_name[ n ] ) ;
                                        if( strlen(s4) > 380 ) {
                                                j = strlen(s4) ;
                                                s4[ j + 1 ] =  0  ;
                                                s4[ j     ] = '+' ;
                                                break ;
                                        }
                                        }
                                sp = & s4[ strlen(s4) ];
                                n++ ; x = x >> 1 ;
                                } // end while
                        }
                        if( local ) {  // local host list
                                        nh++ ;

                                x = host[h].server ;  s2[0] = 0 ; // server X list
                                for(n = 0 ; n < pn_max ; n++) {
                                if( x & 1) strcat(s2,"\tX" ) ;
                                        else strcat(s2,"\t " ) ;
                                        if( strlen( s2 ) > 380 ) break ;
                                        x = x >> 1 ;
                                }

                                x = host[h].client ; s3[0] = 0 ; // client X list
                                for(n = 0 ; n < pn_max ; n++) {
                                if( x & 1) strcat(s3,"\tX" ) ;
                                        else strcat(s3,"\t " ) ;
                                        if( strlen( s3 ) > 380 ) break ;
                                        x = x >> 1 ;
                                }

                        fprintf(web_hosts,"%s\t%u\t%s\t",s1,a0,s4);
                        fprintf(web_hosts,"%u\t%u\t%u", host[h].bytes_in_mx,
host[h].bytes_ot_mx, pu ) ;
                        fprintf(web_hosts,"%s%s\n", s2, s3 ) ; // s2 and s3 start
with \t
                        } // end 'if local'
                        else nf++ ;

                        if((host[h].alerts & ALARM_12) || list_ci || list_traf ||
f_demo) {

                                x = host[h].server ;
                                n = 0 ; sp = s2 ; s2[0] = 0 ;  // s2-server list
                                while(( x > 0) && (n < pn_max)) {
                                if( x & 1) {
                                        sprintf(sp,"%s ", port_name[ n ] ) ;
                                        sp = & s2[ strlen(s2) ];
                                        if( strlen(s2) > 380 ) {
                                                j = strlen(s2) ;
                                                s2[ j + 1 ] =  0  ;
                                                s2[ j     ] = '+' ;
                                                break ;
                                        }
                                        }
                                n++ ; x = x >> 1 ;
                                } // end while

                                for(x = 0 ; x < 10 ; x++ ) {
                                        if(( host[h].s_list[x] == 0) || ( strlen(s2)
> 380 )) break ; //  add ports not in bit map
                                        sprintf(sp,"%u ", host[h].s_list[x] ) ;
                                        sp = & s2[ strlen(s2) ];
                                }

                                x = host[h].client ;
                                n = 0 ; sp = s3 ; s3[0] = 0 ; // s3 - client list
                                while(( x > 0) && (n < pn_max)) {
                                if( x & 1) {
                                        sprintf(sp,"%s ", port_name[ n ] ) ;
                                        sp = & s3[ strlen(s3) ];
                                        if( strlen(s3) > 380 ) {
                                                j = strlen(s3) ;
                                                s3[ j + 1 ] =  0  ;
                                                s3[ j     ] = '+' ;
                                                break ;
                                        }
                                        }
                                n++ ; x = x >> 1 ;
                                } // end while

                                for(x = 0 ; x < 10 ; x++ ) {
                                        if(( host[h].c_list[x] == 0) || ( strlen(s3)
> 380 )) break ; //  add ports not in bit map
                                        sprintf(sp,"%u ", host[h].c_list[x] ) ;
                                        sp = & s3[ strlen(s3) ];
                                }

                        }  //  end build s2,s3
                        if( local ) {             // local host alert list
                                if( list_ci )  {
                                        n0++ ;

fprintf(web_alt0,"%u\t%s\t%u\t%u\t%u\t%s\t%s\t%s\n",

a0,s1,host[h].bytes_in, host[h].bytes_ot,pu,s2,s3,s4);

                                }
                        }
                        if(!(local)) {        // outside host - concern alert list
                                if( list_ci )  {
                                        n1++ ;

fprintf(web_alt1,"%u\t%s\t%u\t%u\t%u\t%s\t%s\t%s\n",

a0,s1,host[h].bytes_in, host[h].bytes_ot,pu,s2,s3,s4);
                                }
                        }

                        if( local ) {                          // local host traffic
alert list
                                if( list_traf ) {
                                        n2++ ;

fprintf(web_traf0,"%u\t%u\t%u\t%u\t%s\t%u\t%s\t%s\t%s\n",(8*traf)/t_diff,(8*host[h].
bytes_in_mx)/t_diff,
                           (8*host[h].bytes_ot_mx)/t_diff,pu,s1,a0,s2,s3,s4) ;
                        }
                  }
                        if(!( local)) {                    // not a local host -
traffic alert list
                                if( list_traf ) {
                                        n3++ ;

fprintf(web_traf1,"%u\t%u\t%u\t%u\t%s\t%u\t%s\t%s\t%s\n",(8*traf)/t_diff,(8*host[h].
bytes_in_mx)/t_diff,
                                  (8*host[h].bytes_ot_mx)/t_diff,pu,s1,a0,s2,s3,s4)
;
                                }
                        } // traffic > x

//   ==============================  ALARMS, OUT OF PROFILE REPORT
==========================
                        if( host[h].alerts & ALARM_2 ) {
                                fprintf( web_alarms, "%u\t%s\tHIGH
CONCERN\t%u\t%u\t%s\n", host[h].alarm_t, s1,a0,traf,s4);// s1=dot-ip
                        } // end ALARM_1

                        if( host[h].alerts & ALARM_1 ) {
                                fprintf( web_alarms,
"%u\t%s\tTouched\t%u\t%u\t%s\n",      host[h].alarm_t, s1,a0,traf,s4);
                        } // end ALARM_0

                        if( host[h].alerts & TRAF_ALARM ) {
                                fprintf( web_alarms, "%u\t%s\tHigh
Traffic\t%u\t%u\t%s\n", host[h].alarm_t, s1,a0,traf,s4);
                        } // end TRAF_ALARM

if(local) {
                  if( profile & 0x10 ) {// profile == 2 or 3, 0=clear every night,
1=save at night, add new
                  register unsigned long s, c ; //  2=alarm & no add, 3 alarm and add
if( f_verbose == 2 ) {printf(" --- Services vs. Profiles\n") ; fflush( stdout ) ; }

                        s = (~host[h].server) & host[h].s_profile ;
                        c = (~host[h].client) & host[h].c_profile ;  // test profile
compliance
                        if( s | c ) {
                                alarm_lines ++ ;
                                fprintf( web_alarms, "%s - Out of Profile.", s1 ) ;

                                if (s ) {
                                        services( s) ;
                                        fprintf( web_alarms, "  Server OoP: %s.",
serstr ) ;

                                        if( c & odd_bit ) {
                                                for(x = 0 ; x < 10 ; x++ ) {
                                                        if( host[h].s_list[x] == 0)
break ; //  add ports not in bit map
                                                        fprintf(web_alarms,"%u ",
host[h].s_list[x] ) ;
                                                }
                                        }
                                }
                                if (c ) {
                                        services( c) ;
                                        fprintf( web_alarms, "  Client OoP: %s.",
serstr ) ;
                                                if( c & odd_bit ) {
                                                for(x = 0 ; x < 10 ; x++ ) {
                                                        if( host[h].c_list[x] == 0)
break ; //  add ports not in bit map
                                                        fprintf(web_alarms,"%u ",
host[h].c_list[x] ) ;
                                                }
                                        }
                                }
                        }
                        fprintf( web_alarms, "\n") ;
                }

                if( profile & 0x01 ) {  // profile == 1 or 3 -> add new bits to
profile
                                                host[h].s_profile |= host[h].server
;
                                                host[h].c_profile |= host[h].client
;  // auto-build profiles
                }
} // end LOCAL_HOST - Profiles

                        if(host[h].bytes_in_pp > host[h].bytes_in_mx)
host[h].bytes_in_mx = host[h].bytes_in_pp ;
                        if(host[h].bytes_ot_pp > host[h].bytes_ot_mx)
host[h].bytes_ot_mx = host[h].bytes_ot_pp ;
                        host[h].bytes_in_pp = host[h].bytes_ot_pp = 0 ;  // zero to
collect bytes for next period

// ==== CLEAR  host[h]  ==== //

                        if( drop_hosts && (!(local)) && (a0 == 0) && ( traf <
host_cut) && (host[h].last < t_cut) ) {

                                if( host[h].up )                    host[ host[h].up
].down = host[h].down ; // if .up not zero
```

```
                                if( host[ host[h].down ].up   == h) host[
host[h].down ].up   = host[h].up ; // one is true
                                if( host[ host[h].down ].root == h) host[
host[h].down ].root = host[h].up ;
                                memset( (void *) & host[h].ip , '\0', (size_t) (
sizeof( struct host_db )  ) );
                                n_host -- ;  n_host_drop ++ ;
                        }

                }  //  .down > 0 confirmed afer lock
        pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
                   //if(!((nh+nf)%100)) printf(".");
                   //if(!((nh+nf)%10000)) printf("\n");
                }  //  .down > 0
  }     // end h++
        file_fail = 8 ;

close_files:
        switch( file_fail ) {
                case 8 :
                        fclose( web_alarms ) ; // fall-thru desired (no 'breaks')
                case 7 :
                        fclose( web_hosts ) ;
                case 6 :
                        fclose( web_traf1 ) ;
                case 5 :
                        fclose( web_traf0 ) ;
                case 4 :
                        fclose( web_alt1 ) ;
                case 3 :
                        fclose( web_alt0 ) ;
                case 2 :
                        fclose( web_info ) ;
                case 1 :
                default:
        }
        if( file_fail <  8 ) printf("  Problem opening a file.  close-index: %u\n",
file_fail ) ;
if( f_verbose == 2) {printf(" WA Web files closed. np:%u at %u\n", np, t_run) ;
fflush( stdout);}


// =========  Determine cut-off value for pruning scans[] ======== //

{
int max_slots, max_print, m ;

max_slots = 0.5 * SCAN_SLOTS ;
max_print = 50 ;

        if( n_scans < max_slots) slots_cut = 0 ;
        else {
                m = 0 ;
                for(i = 14 ; i > 0 ; i-- ) {
                        m += scin[i] ;
                        scans_cut = cix[ i-1 ] ;   // set value of slots_cut to
reduce db
                        if( m > max_slots ) {
                                break ;
        }       }       }

        if( n_scans <50) slots_pr_cut = 0 ;
        else {
                m = 0 ;
                for(i = 14 ; i > 0 ; i-- ) {
                        m += scin[i] ;
                        scans_pr_cut = cix[ i-1 ] ;   // set value of slots_pr_cut
to select for Web
                        if( m > 50 ) {
                                break ;
        }       }       }
}
if( f_verbose == 2 ) {printf(" --- New scans_cut: %u, scans_pr_cut %u\n\n",
 scans_cut, scans_pr_cut ) ; fflush( stdout ) ; }

 {  //  =================  Network Statistics ============== //
        register int m ;
        FILE *net_stats ;


        if( nh < LINES_PER_SCREEN ) web_alert0_0 = web_traf0_0 = 0 ;
        else {
                m = 0 ;
                for(i = 14 ; i > 0 ; i-- ) {
                        m += cin[0][i] ;
                        web_alert0_0 = cix[ i-1 ] ;   // set value of web_alert0_0
                        if( m > LINES_PER_SCREEN ) {
                                break ;
                }       }

                m = 0 ;
                for(i = 14 ; i>0 ; i-- ) {
                        m += tin[0][i] ;
                        web_traf0_0 = tix[ i-1 ] ;   // set value of web_traf0_0
                        if( m > LINES_PER_SCREEN ) {
                                break ;
                }       }
        }


        if( nf < LINES_PER_SCREEN ) web_alert1_0 = web_traf1_0 = 0 ;
        else {
                m = 0 ;
                for(i = 14 ; i>0 ; i-- ) {
                        m += cin[1][i] ;
                        web_alert1_0 = cix[ i-1 ] ;   // set value of web_alert1_0
                        if( m > LINES_PER_SCREEN ) {
                                break ;
                }       }

                m = 0 ;
                for(i = 14 ; i>0 ; i-- ) {
                        web_traf1_0 = tix[ i-1 ] ;   // set value of web_traf1_0
                        m += tin[1][i] ;
                        if( m > LINES_PER_SCREEN ) {
                                break ;
                }       }
        }

        if( n_host > HOST_LIMIT) {
                m = 0 ;
                for(i = 1 ; i < 14 ; i++ ) {
                        host_cut = tix[ i+1 ] ;   // set traf value for host_cut
                        m += tin[1][i] ;
                        if( m > (n_host - HOST_LIMIT) ) {
                                break ;
                }       }
        }

if( f_verbose == 2 ) {printf(" --- Net Stats about to begin.\n") ; fflush( stdout )
; }

        if( (net_stats = fopen("web_stats.txt","wb") )==NULL) {  //  -------OPEN
OUTPUT FILE web_scans.txt
                printf("\nLog-Scans File 'web_stats.txt' Can Not Be Opened\n\n") ;
        }
        else {  //  file 'web_stats' opened OK
                for( m = 1 ; m < 15 ; m++ ) {
                        fprintf(net_stats,"%u\t%u\t%u\t%u\t%u\t%u\t%u\t%u\n",
           cix[m-1],cix[m],cin[0][m],cin[1][m], tix[m-1],tix[m],tin[0][m],tin[1][m]
) ;
                }
        fclose( net_stats ) ;
        }
if( f_verbose == 2) {printf(" Net-stats done. np:%u at %u\n", np, t_run) ; fflush(
stdout);}
}  //  end net-stats

//      if( running == 2 ) break ; // finish up
// // end - while( running )


if( file_fail == 8 ) {
                sleep(1) ; // give file pages time to write
                system(" ./web-prep1.scr >> wp-log.txt 2>> wp-log.txt") ;  // system
call 1 to make Web pages
                sleep(1) ; // give files time to write
                system(" ./web-prep2.scr >> wp-log.txt 2>> wp-log.txt") ;  // system
call 2 to make Web pages
                sleep(1) ; // give files time to write
                system(" ./web-prep3.scr >> wp-log.txt 2>> wp-log.txt") ;  // system
call 3 to make Web pages
                sleep(1) ; // give files time to move
                system(" ./web-prep4.scr >> wp-log.txt 2>> wp-log.txt") ;  // system
call 4 to make Web pages

} else printf("\007 ##########  WEB FILE PROBLEM - NEW WEB PAGES NOT GENERATED
#########\007\n");

printf("%s Web done. Took: %u s, Probes: %u,filed: %u, Alarms: %u %u %u %u\n",
                                    &web_ts[14], time(NULL)-t_2 , n_scans,
n_scan_prt, n0, n1, n2, n3 );
                if( f_verbose || (bt != bt_old) || f_file ) {
                        bt_old = bt ; // do every 15 minutes, not every 5 min
printf("   Pkts: %u, Flows in db: %u, Hosts in db: %u, cut: %u, Host Cut if < %u
byte, age > %u s\n",
                                                        np, n_flow, n_host,
n_host_drop, host_cut, t_here - t_cut);
printf("   Thresholds. CI  for Web-Local Page: %7u, - Web-Outside: %7u, for Alarm:
%7u.\n",
     web_alert0_0, web_alert1_0, *(&web_alert_2) ) ;
printf("   Thresholds. bps for Web-Local Page: %7u, - Web-Outside: %7u, for Alarm:
%7u b/s\n",
    (8* web_traf0_0)/300,  (8*web_traf1_0)/300,  *(&web_traf_2)) ;

printf("   CI Min. for Web.  Inside: %u, Outside : %u",    n0, n1) ;
printf("   Traffic Min.  Inside: %u, Outside : %u\n",  n2, n3) ;
printf("   Local hosts: %u - Foreign Hosts: %u.  Probes Shown:
%u\n\n",nh,nf,n_scan_prt);
```

```
fflush( stdout ) ;
     }

if((f_file) && (running == 1)) sleep(60) ; //delay to read Web
if( f_verbose == 3) {printf(" WA Bottom. np:%u at %u\n", np, t_run) ; fflush(
stdout);}

return ;
}  // end - web_alerts()


void read_thresholds( void ) {
        int h, x ;
        char s[80] ;
        FILE *lc_th_file, *lc_no_alarms_file, *lc_watch_list_file ;

        if((lc_th_file = fopen ("lc_thresholds.txt","rb"))==NULL) {   //  OPEN
lc_thresholds FILE
                printf("File 'lc_thresholds.txt' Can Not Be Opened to Read, t:
%u\n", * (&t_run)) ;
        }
        else {
        register unsigned int a, w, r, p ;
                a = web_alert_2  ;
                w = web_traf_2   ;
                r = restart_hour ;
                p = profile      ;

                while( fscanf(lc_th_file,"%s\t%lu\n", s, &x) != EOF ) {
                        if( ! strcmp( s, "web_alert_2"  ) )  web_alert_2  = x ;
                        if( ! strcmp( s, "web_traf_2"   ) )  web_traf_2   = x ;
                        if( ! strcmp( s, "restart_hour" ) )  restart_hour = x ;
                        if( ! strcmp( s, "profile"      ) )  profile      = x ;
                }

                if(( a != web_alert_2 )||( w != web_traf_2 )||( r != restart_hour
)||( p != profile )) {
                        printf("### NEW VALUES: CI Alarm2: %u, Traf. Alarm: %u
(b/s), Restart: %u, Profile Action: %u\n",

web_alert_2, web_traf_2, restart_hour, profile);
                }
        fclose( lc_th_file ) ;
        }  // end read threshhold file

        if((lc_no_alarms_file = fopen("lc_no_alarms.txt","rb"))==NULL) {   //  OPEN
lc_no_alarms.txt FILE
                printf("File 'lc_no_alarms.txt' Can Not Be Opened to Read, t: %u\n",
* (&t_run)) ;
        }
        else {
        unsigned int a1, a2, a3, a4, h, n_slot ;
                printf(" No-Alarm Set for:" ) ;

                while( fscanf(lc_no_alarms_file,"%u.%u.%u.%u\n", &a1, &a2, &a3, &a4)
!= EOF ) {
                        h = (a1 << 24 ) | (a2 << 16) | (a3 << 8) | a4 ;
                        if(n_slot = find_host( h, 1 ) ) {
                                host[ n_slot ].alerts |= NO_ALARM ;
                                printf("  %u.%u.%u.%u", a1, a2, a3, a4) ;
                        }
                }
        printf("\n") ;
        fclose( lc_no_alarms_file ) ;
        }  // end read threshhold file

        if((lc_watch_list_file = fopen ("lc_watch_list.txt","rb"))==NULL) {   //
OPEN lc_watch.txt FILE
                printf("File 'lc_watch_list.txt' Can Not Be Opened to Read, t:
%u\n", * (&t_run)) ;
        }
        else {
        unsigned int a1, a2, a3, a4, h, n_slot ;
                printf(" Watch_Host Set for:" ) ;

                while( fscanf(lc_watch_list_file,"%u.%u.%u.%u\n", &a1, &a2, &a3,
&a4) != EOF ) {
                        h = (a1 << 24 ) | (a2 << 16) | (a3 << 8) | a4 ;
                        if( n_slot = find_host( h, 1 ) ) {
                                host[ n_slot ].alerts |= WATCH_HOST ;
                                printf("  %u.%u.%u.%u", a1, a2, a3, a4) ;
                        }
                }
        printf("\n") ;
        fclose( lc_watch_list_file ) ;
        }  // end - file opened
return ;
}       // end of read_thresholds()


void services( unsigned int x ) { // puts list of services into serstr
        int n, j ;
        char *ptr ;

        n = 0 ; ptr = serstr ; serstr[0] = 0 ;  // serstr-server list
        while(( x > 0) && (n < pn_max)) {
                if( x & 1) {
                        sprintf(ptr,"%s ", port_name[ n ] ) ;
                        ptr = & serstr[ strlen(serstr) ];
                        if( strlen(serstr) > 380 ) {
                                j = strlen(serstr) ;
                                serstr[ j + 1 ] =  0  ;
                                serstr[ j     ] = '+' ;
                                break ;
                        }
                }
                n++ ; x = x >> 1 ;
        } // end while

        return ;
}  // end services()


#ifdef CHECK_INDEX

unsigned long host_hash( unsigned long ip)
        {// returns host[i] unsigned Index, HOSTS MUST BE LOCKED
        unsigned long x, i, i_mark, n, index, scans_size = HOST_SLOTS ;

        x =  ip ^ (ip <<  6) ;/***  make index from ip ***/
        index = x ;
        for (i = HOST_SHIFT ; i < 32 ; i += HOST_SHIFT ) {
                x = x >> HOST_SHIFT ;
                index ^=  x ;           //  XOR bytes together
        }
        index  = index & HOST_MASK ;  // limit to right HOST_SHIFT bits
        index++ ;               // index can be 1 to 2^n +1

        if( ! pthread_mutex_trylock( &mp_hosts ) ) {
printf(" ### host_hash() without mutex m_host being set. t: %u, Running:
%u\n",*(&t_run),running);
                pthread_mutex_unlock( &mp_hosts ) ;
        }

        if ( index >=  HOST_SLOTS ) {
                printf("index bigger than HOST_SLOTS (%u > %u) at record %u\n",
index, HOST_SLOTS, *(&np));
                exit(-1) ;
        }
        return( index ) ;  //  end - host_hash (ony for CHECK_INDEX
}
#endif


int read_profiles( void ) {  // #############   READ_PROFILES ( & BT[ ] )
##################
        FILE * profile ;
        long check ;
        unsigned long ip, h ;
        int n_pro = 0, n_table = 0, j, n_pts, n_hs ;
        struct stat file_info;
        unsigned char buffer[120] ;

        if( f_verbose == 3) {printf(" Read Profiles Start. np:%u at %u\n", np,
t_run) ; fflush( stdout);}
        if( stat("lc_profiles", &file_info) == 0 ) { // there is a file
                if((profile = fopen("lc_profiles","rb")) !=NULL) {  // and it opened
                        fread( (void *) &check, sizeof( long ), 1, profile) ;

                        // 96 is number of bytes saved per host profile

                        if( check !=  96 ) {
                                printf(" ### File lc_profiles ( %d )does not match
present host db structure size (96) ###\n",
                                                                  check) ;
                        }

                        else {  // check ok
                                register int i ;
pthread_mutex_lock( &mp_hosts) ;  //  ### lock hosts[]
                                while( 1 ) {
                                        if( fread( (void *) &ip, sizeof( long ), 1,
profile) < 1)                   break ;
                                        if( (h = find_host( ip, 1 )) == 0 )
                               break ;
                                        if( fread( (void *) &host[h].s_profile,
sizeof( long  ),     1, profile) < 1 ) break ;
                                        if( fread( (void *) &host[h].c_profile,
sizeof( long  ),     1, profile) < 1 ) break ;
                                        if( fread( (void *) &host[h].s_list[0],
sizeof( short ), 10, profile) < 10) break ;
                                        if( fread( (void *) &host[h].c_list[0],
sizeof( short ), 10, profile) < 10) break ;
                                        if( fread( (void *) &host[h].host_scan[0],
sizeof(long), 5, profile) <  5) break ;
                                        if( fread( (void *) &host[h].scan_cntr[0],
sizeof(char), 4, profile) <  4) break ;
                                        if( fread( (void *) &host[h].port_scan[0],
sizeof(long), 4, profile) <  4) break ;
//if( f_verbose ==  2 ) {
//              printf(" %8x  :%8x %8x :",host[h].ip, host[h].server, host[h].client
) ;
//              for(i=0;i<10;i++) printf(" %u", host[h].s_list[i] ) ;
//              printf(" :");
//              for(i=0;i<10;i++) printf(" %u", host[h].c_list[i] ) ;
//              printf("\n");
//}
                                }   // end while(1)
pthread_mutex_unlock( &mp_hosts) ;  //  ### unlock hosts[]
                        }   // check ok
skip_profiles:
                        fclose( profile ) ;
                }  //  file opened ok
        } // stat -> end file-ok
//      if( f_verbose == 3) {printf(" Read Profiles Mid. np:%u at %u\n", np, t_run)
; fflush( stdout);}

        if( stat("lc_traf_table", &file_info) == 0 ) { // there is a file
                if((profile = fopen("lc_traf_table","rb")) != NULL) {
                        fread( (void *) &check, sizeof( long ), 1, profile) ;
                        if( check !=  sizeof( struct byte_table ) ) { // byte table
may change size due to revision
                                printf(" ### File lc_traf_table ( %d )does not match
present bs[]. structure size! ###\n",
                                                check) ;
                        }
                        else {  // check is ok
                                for( j = (TRAFFIC_VALUES - 1) ; j >= 0 ; j-- ) {
                                        if( fread( (void *) &bs[j].t, check      ,
1, profile) < 1 ) break ;
                                        n_table++ ;
//if((f_verbose == 2) && bs[j].t) printf("j:%2d  %5d  %8d %8d %8d %8d\n",j,bs[j].t,
bs[j].bytes_in,
//                                                              bs[j].bytes_out,
bs[j].bytes_loc, bs[j].bytes_oo ) ;
                                }
                        }   // end - 'check' is right
                        fclose( profile ) ;
                }  // end - file opened ok
        }  // end - there is a file
        bt = TRAFFIC_VALUES - 1 ;
        printf("  Read - Local Host Profiles: %d, Traf.Table Lines: %d\n", n_pro,
n_table ) ;                     // call save_profiles
        printf("                       Active Locals: %d, n_host: %d\n", active_locals,
n_host  ) ;                     // call save_profiles
        if( f_verbose == 3) {printf(" Read Profiles End. np:%u at %u\n", np, t_run)
; fflush( stdout);}
        return( n_pro ) ;
}  // end read_profiles


int save_profiles( void ) {  // #############   SAVE_PROFILES ( & BT[ ] )
##################
        FILE * profile ;
        long check ;
        unsigned long ip, hx ;
        int m, n_pro = 0, n_table=0, i, j ;
if( f_verbose == 2) {printf(" Save Profiles Start. np:%u at %u\n", np, t_run) ;
fflush( stdout);}
```

```
    if((profile = fopen("lc_profiles","wb")) != NULL) {
        check = 96 ;
        fwrite( (void *) &check, sizeof( long ), 1, profile) ;

        for( hx = 1 ; hx < HOST_SLOTS ; hx++ ) {  // save profiles of local
host
            if( host[hx].alerts & 0x1 ) {
    pthread_mutex_lock( &mp_hosts) ;  //  ###  lock hosts[]
                if( fwrite( (void *) &host[hx].ip,        sizeof(
long  ),  1, profile)  < 1 ) break ;
                if( fwrite( (void *) &host[hx].s_profile, sizeof(
long  ),  1, profile)  < 1 ) break ;
                if( fwrite( (void *) &host[hx].c_profile, sizeof(
long  ),  1, profile)  < 1 ) break ;
                if( fwrite( (void *) &host[hx].s_list[0], sizeof(
short ), 10, profile)  < 10) break ;
                if( fwrite( (void *) &host[hx].c_list[0], sizeof(
short ), 10, profile)  < 10) break ;
                if( fwrite( (void *) &host[hx].host_scan[0],
sizeof(long),   5, profile) <  5) break ;
                if( fwrite( (void *) &host[hx].scan_cntr[0],
sizeof(char),   4, profile) <  4) break ;
                if( fwrite( (void *) &host[hx].port_scan[0],
sizeof(long),   4, profile) <  4) break ;
                n_pro++ ;
//              if( f_verbose ==  2 ) {
//                      printf(" %8x  :%8x %8x
:",host[hx].ip, host[hx].server, host[hx].client ) ;
//                      for(i=0;i<10;i++) printf(" %u.",
host[hx].s_list[i] ) ;
//                      printf(" :");
//                      for(i=0;i<10;i++) printf(" %u.",
host[hx].c_list[i] ) ;
//                      printf("\n");
//                  }
        pthread_mutex_unlock( &mp_hosts) ;  //  ###  unlock hosts[]
            }
        } // end for(hx)
        fclose( profile ) ;
    }
    if( f_verbose == 3) {printf(" Save-Profiles Mid. np:%u at %u\n", np, t_run)
; fflush( stdout);}

    if((profile = fopen("lc_traf_table","wb"))!=NULL) {
pthread_mutex_lock( &mp_bs) ;  //  ### lock bs[]
        check = sizeof( struct byte_table ) ;
        fwrite( (void *) &check, sizeof( long ), 1, profile) ;
        m = bt  ;
        m++ ;
        if( m > TRAFFIC_VALUES) m -= TRAFFIC_VALUES ;  // prevents thread
sync problem
        for( j = 0 ; j < TRAFFIC_VALUES ; j++ ) {  // j is counter, bs.[bt +
1] not reliable in this thread
            m-- ;
            if(m < 0 ) m += TRAFFIC_VALUES ;  // m is index
//          if( bs[m].t == 0) break ;
            if( fwrite( (void *) &bs[m].t, sizeof( struct byte_table ),
1, profile) < 1 ) break ;
            n_table++ ;
//if((f_verbose == 2)&&(j==0)) printf("size:%d m:%2d  %11d  %8d %8d %8d %8d\n",
sizeof( struct byte_table ),
//                  m,bs[m].t, bs[m].bytes_in, bs[m].bytes_out,
bs[m].bytes_loc, bs[m].bytes_oo ) ;
```

```
                }
                fclose( profile ) ; // break comes here
pthread_mutex_unlock( &mp_bs) ;  //  ### unlock bs[]
        }
        printf("  Saved (at %u s) - Local Host Profiles: %d, Traf.Table Lines:
%d\n", t_run, n_pro, n_table ) ;
        if( f_verbose == 3) {printf(" Save Profiles End. np:%u at %u\n", np, t_run)
; fflush( stdout);}
        return( n_pro ) ;
}  // end write_profiles



void record_probe( unsigned long hs, unsigned long d_ip, unsigned short des_port ) {
// sets bit map for ip-scan and port-scan detection
        register int x1, unlock = 0 ;  // mutex mp_hosts should be locked before
calling
  register unsigned int m4, m2 ;

        if(( f_verbose == 2) || (f_verbose == 3 )) {
                printf(" Record-Probe Start. np:%u at %u, hs: %u, d_ip: %8x, port:
%u\n",np,*(&t_run),hs,d_ip,des_port);
                fflush( stdout);
        }

        if(f_verbose == 2) printf(" ### SCAN hs: %u, da :%8x, ",hs,d_ip ) ;

        x1 =     (d_ip>>5) & 0x3 ;  // index 0 to 3
        m2 = (unsigned int) 1 << ( d_ip & 0x001f) ; // 1 bit set in 32-bit field

        if( ! (host[ hs ].host_scan[ x1 ] & m2) ) { //  ----- check hscan128 bit
                host[ hs ].host_scan[ x1 ]  |= m2 ;   //  --- set bit for hscan128
          host[hs].scan_cntr[1] ++ ;
        }

        m4 = (unsigned int) 1 << ( ( (d_ip>>24) ^ (d_ip>>16) ) & 0x001f ) ; // 1 bit
set in 32-bit field

        if( ! (host[ hs ].host_scan[ 4 ] & m4) ) { //  ----- check hscan32 bit
                host[ hs ].host_scan[ 4 ] |= m4 ; //  set bit for hscan32
          host[hs].scan_cntr[0] ++ ;

                if( host[hs].scan_cntr[0] > 24 ) {  //  clear once hscan32 is nearly
full
                        host[hs].host_scan[0] = host[hs].host_scan[1] = 0 ;
                        host[hs].host_scan[2] = host[hs].host_scan[3] =
host[hs].host_scan[4] = 0 ;
                        host[hs].scan_cntr[0]  = host[hs].scan_cntr[1]  = 0 ;
                }
        }

        if( host[hs].scan_cntr[0] < (((3*host[hs].scan_cntr[1]) >> 3) - 2) ) { //
hscan32 < (3/8)hscan128 - 1
                host[hs].alerts  |= IP_SCAN ;
                host[hs].concern += HOST_SCAN_CI ;
                host[hs].host_scan[0] = host[hs].host_scan[1] = 0 ;
                host[hs].host_scan[2] = host[hs].host_scan[3] =
host[hs].host_scan[4] = 0 ;
                host[hs].scan_cntr[0]  = host[hs].scan_cntr[1]  = 0 ;

                if( f_verbose == 2 ) printf(" ###  BINGO  p32: %u, p128:
%u\n",host[hs].scan_cntr[0],host[hs].scan_cntr[1]);
        }
        else {
                if(( f_verbose == 2 ) && ( host[hs].scan_cntr[1] > 2 )) {
                        printf(" --- NOGO h32: %u, h128: %u x32:%8x,
x128:%8x,%8x,%8x,%8x  x1:%d, x2:%8x\n",
                        host[hs].scan_cntr[0],  host[hs].scan_cntr[1],
host[hs].host_scan[4],
                        host[hs].host_scan[0], host[hs].host_scan[1],
host[hs].host_scan[2], host[hs].host_scan[3], x1,m2);
        } }

        if( des_port < 1024 ) {  // avoid false positives due to Napster
                register unsigned long pp, pm, pi ; //  ##########  PORT SCAN
DETECTION ########## //
                pp = des_port &0x1f                ; // 0 to 31
                pi = (des_port >> 5) & 1           ; // 0 or 1
                pm = (unsigned long) 1 <<  pp      ; // 1 '1' in 32-bit field

                if(! (pm & host[ hs ].port_scan[ pi ])) { // new long-term bit to
set ?

                        host[ hs ].port_scan[ pi ] |= pm ;  // used to detect
short-term port scans
                        host[hs].scan_cntr[2] ++ ;

                        if( host[hs].scan_cntr[2] > PORT_SCAN_MAX ) {
                                host[hs].concern +=  PORT_SCAN_CI ;
                                host[hs].alerts |= PT_SCAN_ALERT ;
                                host[hs].port_scan[0] = host[hs].port_scan[1] = 0 ;
// clear short-term port-scan bits & counter
                                host[hs].scan_cntr[2]  = 0 ;
if(f_verbose == 2) printf(" ### Port Scan - Short-Term Detected. hs:%u\n",hs ) ;
                        }
                } // end - ling-term threshold exceeded?
                else {
                        if(f_verbose==2) {
                                printf("Port %u - ST bit already set. pi:%u, pp: %u,
pm: %8x, ST: %8x,%8x,LT: %8x-%8x\n\n",des_port, pi,pp,pm,
                                host[hs].port_scan[0], host[hs].port_scan[1],
host[hs].port_scan[2], host[hs].port_scan[3]) ;
                                fflush( stdout ) ;
                        }
                }

                if(! (pm & host[ hs ].port_scan[ pi | 2 ])) { // pi|2 = 2 or 3, new
long-term bit to set ?

                        host[ hs ].port_scan[ pi | 2 ] |= pm ;  // used to detect
long-term  port scans
                        host[hs].scan_cntr[3] ++ ;

                        if( host[hs].scan_cntr[3] > PORT_SCAN_LT_MAX ) {
                                host[hs].concern +=  PORT_SCAN_LT_CI ;
                                host[hs].alerts |= LT_PT_SCAN ;
                                host[hs].port_scan[2] = host[hs].port_scan[3] = 0 ;
// reset long-term port-scan bit map & counter
                                host[hs].scan_cntr[3]  = 0 ;
if(f_verbose==2) printf(" ### Port Scan -  Long-Term Detected. hs:%u\n",hs ) ;
                        }
                } // end - ling-term threshold exceeded?
                else {
                        if(f_verbose==2) {
                                printf("Port %u - LT bit already set. pi:%u, pp: %u,
pm: %8x, ST: %8x,%8x,LT: %8x-%8x\n\n",des_port, pi,pp,pm,
                                host[hs].port_scan[0], host[hs].port_scan[1],
host[hs].port_scan[2], host[hs].port_scan[3]) ;
                                fflush( stdout ) ;
                        }
                }

                if( f_verbose == 3) {
                        printf(" Record-Probe End. np:%u at %u, hs: %u, d_ip: %8x,
port: %u\n",
                         np, *(&t_run) ,hs, d_ip, des_port);
                        fflush( stdout);
                }
        } // end - port scan check
        return ;
} // - end - record_probe()


void  suicide( int yesterday ) {  //  kill this instance of lancope, start fresh
        char command[50] ;

        //  yesterday = (t_here % 86400) % 30 ;// day of month 0-29 - name of
directories for saving files
        setuid( 0 ) ;
        sprintf(command,"lc_restart.sh %d 1>> lc_log.txt 2>>lc_log.txt &",
yesterday) ;
        printf("system call : %s\n", command) ;
        if( system( command ) ) {  // returns zero if ok
                printf(" #### FAILED ####\n" ) ;
                return ;
        }
        printf(" #### RESTART CALL OK, SHUTTING DOWN NOW #### LOGS SAVED ON '%d'\n",
yesterday);
        running = 2 ;  // end other threads if 'lc_restart' started.
        kill_tcpdump() ;
        return ;
} // end - suicide()


void userhandler(int sig) {  // SIGTERM handler (kill <PID> response)
        int i ;
                i = save_profiles() ;
                printf("\n ######### SIGTERM (%d) - Saved %u Profiles.  No Restart.
###########\n\n", sig, i ) ;
                running = 2 ;
                kill_tcpdump() ;
                return ;
} // end - userhandler

// ###  END OF FILE ####
```